TUGAS AKHIR – KI141502

Analisis Performa *Proactive Routing Protocol* DSDV dan OLSR pada *Vehicular Ad hoc Network* (VANET) Menggunakan NS-3

MADE DIA AGUSTYA
NRP 5111100037

Dosen Pembimbing I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Hudan Studiawan, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2015

FINAL PROJECT – KI141502

Performance Analysis of Proactive Routing Protocol DSDV and OLSR on *Vehicular Ad hoc Network* (VANET) Using NS-3

MADE DIA AGUSTYA
NRP 5111100037

Advisor I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Advisor II
Hudan Studiawan, S.Kom., M.Kom.

DEPARTEMENT OF INFORMATICS ENGINEERING
Faculty of Information Technology
Sepuluh Nopember Intitute of Technology
Surabaya, 2015

LEMBAR PENGESAHAN

Analisis Performa *Proactive Routing Protocol* DSDV dan OLSR pada *Vehicular Ad hoc Network* (VANET) Menggunakan NS-3

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
Pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
MADE DIA AGUSTYA
NRP: 5111 100 037

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
   NIP: 19841016 200812 1 002 (Pembimbing 1)

2. Hudan Studiawan, S.Kom., M.Kom.
   NIP: 19870511 201212 1 003 (Pembimbing 2)

SURABAYA
JUNI 2015

Analisis Performa *Proactive Routing Protocol* DSDV dan OLSR pada *Vehicular Ad hoc Network* (VANET) Menggunakan NS-3

Nama Mahasiswa : Made Dia Agustya
NRP : 5111100037
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.
Dosen Pembimbing 2 : Hudan Studiawan, S.Kom., M.Kom.

ABSTRAK

Vehicular Ad Hoc Network (VANET) merupakan turunan dari MANET adalah inovasi baru dalam teknologi yang membantu kebutuhan manusia dalam berkomunikasi. VANET dapat mendukung komunikasi langsung antara kendaraan (vehicle to vehicle) dengan adanya infrastruktur jaringan nirkabel. Dengan berbasis ad hoc VANET juga memungkinkan terjadinya komunikasi pada node yang ditangani oleh protokol untuk mencari rute jaringan. Namun, dengan penggunaan protokol hasil adaptasi dari MANET pada VANET sehingga belum bekerja dengan baik sehingga perlu dilakukan analisis kinerja protokol. Pada Tugas Akhir ini yang diteliti yaitu protokol proaktif MANET jenis DSDV dan OLSR. Penelitian ini menggunakan NS-3 sebagai simulator VANET.

Protokol diujikan ke dalam skenario dengan peta berbentuk grid dan peta riil Surabaya dengan memvariasikan jumlah kendaraan simulasi. Masing-masing skenario menunjukkan performa yang hampir sama. Performa protokol DSDV lebih rendah pada packet delivery ratio, kurang stabil pada delay dan tingginya jumlah routing overhead ketika jumlah kendaraan bertamabah. Sementara protokol OLSR menunjukkan performa yang lebih tinggi dan pada packet delivery ratio, stabil pada delay, dan rendah pada routing overhead.

Kata kunci: VANET, MANET, DSDV, OLSR, NS-3.

Analisis Performa *Proactive Routing Protocol* DSDV dan OLSR pada *Vehicular Ad hoc Network* (VANET) Menggunakan NS-3

Student's Name	**: Made Dia Agustya**
Student's ID	**: 5111100037**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**: Hudan Studiawan, S.Kom., M.Kom.**

ABSTRACT

Vehicular Ad Hoc Network (VANET), a sub-class of MANET, is a new innovation technology that helps human need to communicate. VANET can support direct communication between vehicles with their wireless network infrastructure. VANET with ad hoc based also enables communication on a node that is handled by the protocol to find network route. However, with the use of adaptation from MANET protocols in VANET so that has not worked well and it is necessary to analyze the performance of the protocol. In this project were studied proactive routing protocol of MANET DSDV and OLSR type. This study uses NS-3 as a simulator of VANET.

The protocol was tested in a scenario with a map-shaped grid and map real Surabaya by varying the number of vehicle simulation. Each of these scenarios show almost the same performance. DSDV has lower protocol performance in packet delivery ratio, the less stable the delay and the high number of routing overhead when the number of vehicles increase. While the protocol OLSR showed higher performance and the packet delivery ratio, steady on delay, and low in routing overhead.

Keywords: VANET, MANET, DSDV, OLSR, NS-3.

KATA PENGANTAR

Puji syukur kepada Ida Sang Hyang Widhi Wasa, yang telah melimpahkan berkat dan rahmatnya sehingga penulis bisa menyelesaikan Tugas Akhir yang berjudul "**Analisis Performa *Proactive Routing Protocol* DSDV dan OLSR pada *Vehicular Ad hoc Network* (VANET) Menggunakan NS-3**" dengan baik dan tepat waktu.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan-bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Ida Sang Hyang Widhi Wasa, Tuhan Yang Maha Esa atas limpahan berkat dan rahmat-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
2. Kedua orang tua penulis, Bapak Nengah Widiada dan Ibu Made Waluyati, adik saya Nyoman Triska Ariyanti, atas semangat, motivasi dan dukungannya kepada penulis, sehingga penulis dapat termotivasi untuk mengerjakan Tugas Akhir ini.
3. Bapak Dr.Eng. Radityo Anggoro S.Kom., M.Sc. selaku dosen pembimbing pertama, yang telah memberikan bimbingan, dukungan, masukkan, serta semua yang telah diberikan kepada penulis, sehingga Tugas Akhir ini bisa selesai dengan tepat waktu.
4. Bapak Hudan Studiawan, S.Kom., M.Kom. selaku dosen pembimbing kedua, atas bimbingan, arahan, bantuan, ide serta semua yang telah diberikan kepada penulis untuk menyelesaikan Tugas Akhir ini.
5. Putu Candra Yuni Artini, atas perhatian spesialnya dengan semangat, motivasi dan dukungannya kepada penulis, sehingga penulis dan beliau sama-sama termotivasi untuk mencapai tahap Sarjana.

6. Ibu Nanik Suciati, S.Kom., M.Kom., Dr.Eng. selaku ketua jurusan Teknik Informatika ITS, Ibu Bilqis , S.Kom., M.Kom. selaku dosen wali penulis, dan segenap dosen Teknik Informatika yang telah memberikan ilmunya kepada penulis.
7. Pak Yudi, Pak Sugeng, Mas Jumali dan segenap staf Tata Usaha yang telah memberikan segala bantuan dan kemudahan kepada penulis selama menjalani kuliah di Teknik Informatika ITS.
8. Teman-teman seperjuangan Tugas Akhir di Lab AJK, Rickson, Prana, Harum, Desta, Dimas, dan Vivi.
9. Seluruh teman Teknik Informatika ITS angkatan 2011, yang menjadi teman seperjuangan penulis.
10. Teman-teman TPKH TC yang telah memberikan bantuan.
11. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya Tugas Akhir ini.

Kesempurnaan tentu sangat jauh tercapai pada Tugas Akhir ini, maka penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Juni 2015

MADE DIA AGUSTYA

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

Bab ini membahas garis besar penyusunan Tugas Akhir yang meliputi latar belakang, tujuan pembuatan, rumusan dan batasan permasalahan, metodologi penyusunan Tugas Akhir, dan sistematika penulisan.

1.1 Latar Belakang

Teknologi Informasi dan Komunikasi atau biasa disebut *Information Communication Technology* (ICT) telah banyak membantu produktivitas manusia yang semakin meningkat. Salah satu inovasi teknologi yang membantu manusia dalam berkomunikasi dengan mudah adalah *Vehicular Ad hoc Network* (VANET). VANET termasuk dalam jaringan komunikasi nirkabel dan merupakan turunan dari *Mobile Ad hoc Network* (MANET). VANET mendukung komunikasi langsung antar kendaraan (*vehicle to vehicle*) dengan adanya infrastruktur jaringan nirkabel. VANET dengan basis *ad hoc* memungkinkan terjadinya komunikasi pada simpul (*node*) yang ditangani oleh protokol untuk mencari rute jaringan. Oleh karena hal itu, kendaraan tetap bisa berkomunikasi walaupun sedang berpindah secara tidak tetap.

Protokol pada jaringan *ad hoc* terdiri dari 2 (dua) tipe yaitu protokol proaktif dan protokol reaktif. Pada Tugas Akhir ini difokuskan pada penelitian terhadap protokol *ad hoc* tipe proaktif. Terdapat dua jenis protokol proaktif, yaitu *Destination-Sequenced Distance Vector* (DSDV) dan *Optimized Link State Routing* (OLSR). DSDV merupakan jenis protokol yang melakukan *routing* dengan metode turunan dari algoritma *Bellman-Ford*, sedangkan OLSR merupakan protokol *routing* untuk IP yang telah mengalami optimasi pada jaringan *mobile ad hoc* dan juga *wireless ad hoc*. Namun kedua protokol tersebut pada dasarnya dibuat untuk MANET sehingga belum dapat bekerja dengan baik pada VANET.

Untuk itu diperlukan analisis performa pada kedua protokol *routing* proaktif tersebut.

Implementasi VANET di dunia nyata membutuhkan biaya yang sangat mahal dan jangkauan yang luas serta tidak memungkinkan untuk diimplementasikan secara langsung. Maka dari itu penulis menggunakan cara simulasi agar dapat menjalankan sistem dengan baik dan sesuai dengan lalu lintas di dunia yaitu menggunakan NS-3 sebagai simulator dan SUMO dibantu dengan teknologi OpenStreetMap untuk membentuk mobilitas *vehicular*. Hal yang dianalisis dari uji coba untuk menunjukkan performa diukur melalui perhitungan *packet delivery ratio* (PDR), *end-to-end delay* (E2D), dan *routing overhead* (RO). Dan hasil yang diharapkan dari tugas akhir ini adalah suatu kesimpulan tentang hal-hal yang mempengaruhi performa protokol *routing* jenis proaktif yang dapat dimodifikasi agar protokol *routing* mampu bersifat adaptif terhadap lingkungan VANET.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Mengapa protokol *routing* DSDV dan OLSR bisa berjalan dengan baik pada lingkungan MANET sedangkan pada VANET tidak?
2. Bagaimana performa protokol DSDV dan OLSR pada lingkungan VANET dalam bentuk peta grid dan peta riil?
3. Bagaimana pengaruh perubahan jumlah node terhadap performa protokol dan OLSR pada VANET?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, diantaranya sebagai berikut:

1. Protokol *routing* yang diimplementasikan adalah protokol DSDV dan OLSR.
2. Uji coba dan simulasi menggunakan NS-3 (*Network Simulator* 3).
3. Skenario uji coba VANET dibuat menggunakan SUMO.

1.4 Tujuan dan Manfaat

Tugas akhir ini bertujuan untuk menganalisis performa protokol *routing* proaktif DSDV dan OLSR pada lingkungan VANET.

Manfaat yang ada pada pembuatan tugas akhir ini yaitu sebagai bahan dasar untuk penelitian protokol *routing* proaktif DSDV dan OLSR agar dapat bersifat adaptif pada jaringan VANET.

1.5 Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal tersebut dijelaskan secara garis besar tentang alur pembuatan sistem.
2. Studi literatur
   Pada tahap ini dilakukan studi literatur mengenai tools dan metode yang digunakan. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal dan dokumentasi dari internet.
3. Implementasi protokol *routing*
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.
4. Uji coba dan evaluasi
   Pada tahapan ini dilakukan uji coba terhadap aplikasi yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem,

mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

5. Penyusunan buku tugas akhir.

   Pada tahapan ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir.

1.6 Sistematika Penulisan

Buku tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

BAB II. TINJAUAN PUSTAKA

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang untuk mendukung pembuatan tugas akhir ini.

BAB III. PERANCANGAN

Bab ini berisi perancangan metode yang nantinya akan diimplementasikan dan dilakukan uji coba.

BAB IV. IMPLEMENTASI

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa implementasi skenario mobilitas *vehicular*, konfigurasi sistem dan skrip analisis yang digunakan untuk menguji performa protokol *routing*.

BAB V. UJI COBA DAN EVALUASI

Bab ini menjelaskan tahap pengujian sistem dan pengujian performa dalam skenario mobilitas *vehicular* yang dibuat.

BAB VI. PENUTUP

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan terhadap rumusan masalah yang ada dan saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan pengimplementasian perangkat lunak. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap protokol *routing*, alat, serta definisi yang digunakan dalam pembuatan Tugas Akhir.

2.1. *Vehicle Ad hoc Network* (VANET)

Vehicular Ad Hoc Network (VANET) merupakan kategori khusus dari *mobile ad hoc networks* (MANET's), ditandai dengan mobilitas tinggi dan konektivitas yang rendah. VANET adalah sebuah teknologi baru yang memadukan kemampuan komunikasi nirkabel kendaraan menjadi sebuah jaringan yang bebas infrastruktur [1].

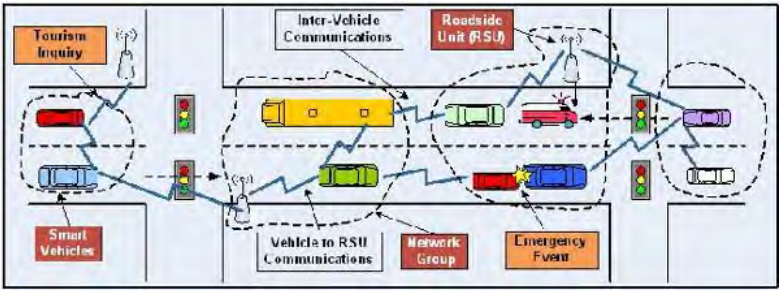
Perbedaan mendasar dari VANET dibandingkan dengan MANET adalah pergerakan *node* dengan kecepatan yang sangat tinggi, mobilitas yang memiliki jalur tertentu, media penyimpanan yang mencukupi, sumber daya yang terpenuhi kepadatan *node* yang tidak dapat diprediksi, dan lingkungan komunikasi yang sulit karena jangka waktu yang sangat singkat [1].

Dengan adanya perbedaan-perbedaan tersebut, munculah kesulitan-kesulitan dalam mengadaptasi protokol-protokol *routing* pada MANET untuk diterapkan pada topologi VANET. Tantangan mendasar dalam VANET adalah (i) bagaimana menyediakan koneksi yang stabil untuk pengguna jaringan ketika berada di jalan raya dan (ii) bagaimana cara mengefisienkan komunikasi antar kendaraan *(vehicle-to-vehicle)* atau pun komunikasi antara kendaraan dengan infrastruktur sepanjang jalan *(road-infrastructur-to-vehicle)* [1].

Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui komunikasi V2V *(Vehicle to Vehicle)* dan V2I *(Vehicle to Infrastructure)*, bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara

menyediakan pengemudi dan penumpang dengan akses internet. Dalam VANET, RSU *(Roadside Unit)* dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan melakukan *broadcast* pesan yang terkait seperti pemberitahuan untuk memberikan pengendara informasi [2]. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas melalui komunikasi V2I, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi, dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan.

Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan awal untuk menanggapi situasi yang abnormal. Untuk mencapai hal ini, OBU *(On-Boards Unit)* secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan, status rem, sudut kemudi, lampu sen, percepatan / perlambatan, kondisi lalu lintas [2].



Gambar 2-1 Skema VANET

Pada tugas akhir ini VANET digunakan sebagai jenis jaringan nirkabel dalam menganalisis kinerja protokol *routing* DSDV dan OLSR.

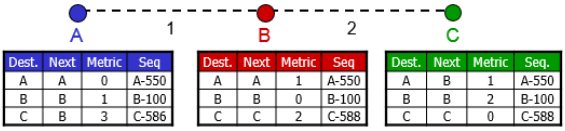
2.2. Protokol *Routing* Proaktif MANET

Mobile Ad Hoc Network (MANET) adalah sekumpulan *mobile node* yang terdesentralisasi yang mana proses pertukaran informasinya melalui media transmisi nirkabel atau *wireless*. Topologi jaringan MANET tidak terstruktur dimana tiap *node* bisa masuk dan keluar dari jaringan sekehendaknya. Tiap *node* bisa berkomunikasi dengan *node* lainnya dalam jangkauan transmisi tertentu. Untuk komunikasi di luar jangkauan, suatu node membutuhkan bantuan node yang lain yang bertindak sebagai *"bridge"* sehingga *node* dalam MANET bisa bertindak sebagai terminal dan router [3]. Protokol *routing* MANET jenis proaktif memelihara informasi *routing* dari semua *node* dalam jaringan melalui menambahkan rute baru atau memperbaharui rute yang telah ada dengan cara mendistribusikan informasi *routing* ke sekitarnya secara periodik. Kelebihan dari hal tersebut yaitu tersedianya semua rute dari tujuan ke destinasi manapun ketika dibutuhkan. Untuk jaringan dinamis dengan skala besar, konvergensi mungkin tidak dapat dilakukan. Tabel *routing* akan semakin bertambah dikarenakan ukuran peta bertambah besar atau jumlah *node* dalam jaringan bertambah banyak, buka dikarenakan jumlah rute yang benar-benar diperlukan [4]. Dua jenis protokol proaktif MANET yang diimplementasikan dalam NS-3 adalah DSDV dan OLSR.

2.2.1. *Dynamic-Sequenced Distances Vector* (DSDV)

Destination-Sequenced Distance Vector (DSDV) merupakan protokol *routing* yang menggunakan algoritma *Distance-Vector* dan algoritma *shortest path Bellman-Ford*. Mekanisme DSDV dalam menemukan rute di dalam *mobile ad hoc network* (MANET) berbeda. *Routing table* yang digunakan protokol ini menyimpan *hop* (loncatan) selanjutnya dari *node* awal, *cost* dari *node* awal ke *node* tujuan, serta *destination sequence number* yang berasal dari node tujuan. Pada dasarnya algoritma *Distance-Vector* tidak bebas pengulangan *(loop free)*, oleh karena itu *destination sequence number* digunakan supaya tidak terjadi *looping* dalam proses *routing*. *Destination sequence number* juga berguna untuk menjaga

informasi *routing table* supaya menjadi informasi yang terbaru dengan memperbaharui rute lama menjadi rute yang baru.



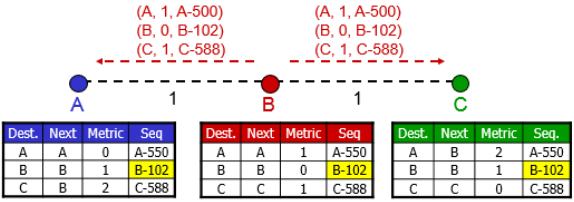
Dest.	Next	Metric	Seq
A	A	0	A-550
B	B	1	B-100
C	B	3	C-586

Dest.	Next	Metric	Seq
A	A	1	A-550
B	B	0	B-100
C	C	2	C-588

Dest.	Next	Metric	Seq.
A	B	1	A-550
B	B	2	B-100
C	C	0	C-588

Gambar 2-2 Tabel routing tiap node

Pada Gambar 2-2, setiap *node* mencatat *destination* (tujuan) yang mungkin tercapai, *next node* yang mengarah ke *destination, cost (metric)*, dan *sequence number*. Setiap *node* saling bertukar informasi secara rutin dengan melakukan *broadcast* ke *node* tetangga (*neighbor node*). Pembaruan *routing table* juga bisa terjadi apabila ada *event* tertentu, seperti rute putus atau pergerakkan *node* yang menyebabkan perubahan topologi jaringan dan perubahan informasi pada tabel.



Dest.	Next	Metric	Seq
A	A	0	A-550
B	B	1	B-102
C	B	2	C-588

Dest.	Next	Metric	Seq
A	A	1	A-550
B	B	0	B-102
C	C	1	C-588

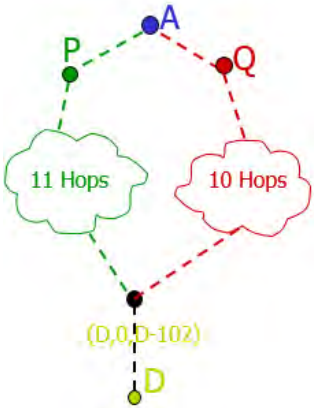
Dest.	Next	Metric	Seq.
A	B	2	A-550
B	B	1	B-102
C	C	0	C-588

Gambar 2-3 Node B melakukan broadcast

Pada Gambar 2-3, *node* B melakukan *increment* terhadap *sequence number* menjadi 102 dan melakukan *broadcast* informasi *routing table* baru ke *node-node* tetangganya (A dan C). *Broadcast* ini akan terus dilakukan selama masih ada *node* pada jaringan yang terhubung sehingga *routing table* selalu baru.

Pada Gambar 2-4, *node* D merupakan source *node*, sedangkan *node* A merupakan *destination node*. Karena adanya *routing table*, maka D dapat dengan mudah mengidentifikasi rute

dengan *cost* atau *hops* (loncatan) terpendek yaitu melalui Q. *Node* D melakukan *broadcast* dengan *sequence number* D-102 melalui P dan Q. Informasi sampai ke P dan Q dengan *sequence number* 102 dengan jumlah *hop* 14 di Q dan 15 di P. *Routing table* yang dipilih adalah routing table dengan *sequence number* terbesar dan jumlah *hop* terkecil. Oleh karena *sequence number* di P dan Q sama-sama 102, maka informasi yang akan diteruskan ke *node* A adalah informasi *routing table* dari *node* Q dengan jumlah *hop* 14.



Gambar 2-4 Node D mengirim paket ke node A melalui 2 jalur dengan hop berbeda

Pembaharuan rute pada protokol *routing* DSDV bersifat *time-driven* (periodik) ataupun *event-driven* (digerakkan oleh fenomena tertentu). Setiap *node* bertukar informasi dengan *node-node* tetangganya secara periodik untuk memperoleh informasi *routing table* yang terbaru. Saat terjadi perubahan signifikan tertentu dari *update* terakhir, suatu *node* dapat mengirim informasi dari *routing table* yang telah berubah dengan digerakkan oleh *trigger* atau *event* tertentu.

DSDV memiliki dua cara saat memperbaharui *routing table*. Pertama adalah *full-dump* yang memperbarui seluruh isi *routing table*. *Incremental update*, merupakan cara lain yang hanya

memuat informasi yang berubah sejak pembaharuan terakhir. *Incremental update* dapat dikirim dengan satu NDPU *(Network Data Packet Unit)* sedangkan *full-dump* dikirim dengan menggunakan beberapa NDPU.

DSDV merupakan protokol *routing* yang efisien. Dengan adanya *sequence number*, DSDV bebas dari pengulangan (*loop free*). *Delay* (keterlambatan) untuk penemuan rute baru juga relatif rendah karena saat dibutuhkan *destination* yang baru, *source node* telah menyimpan rute dari *source* ke *destination* di dalam *routing table* yang diperbarui secara rutin.

Sebagai protokol proaktif, DSDV perlu memperbaharui *routing table* secara rutin sehingga mengkonsumsi banyak energi baterai dan *bandwith* meskipun jaringan tersebut sedang dalam kondisi *idle*. Akibatnya protokol ini kurang cocok untuk jaringan dengan jumlah *node* yang sangat besar. Saat topologi jaringan berubah, *sequence number* baru dibutuhkan. DSDV tidak stabil hingga perubahan *routing table* tersebar di seluruh *node* pada jaringan. Karena alasan ini DSDV tidak tepat digunakan pada jaringan dengan mobilitas tinggi.

DSDV efektif untuk jaringan *ad-hoc* dengan populasi rendah karena perubahan topologi juga relatif rendah. Jaringan dengan populasi tinggi namun frekuensi perubahan topologi yang rendah juga baik untuk protokol DSDV. Hal ini disebabkan karena tidak dibutuhkan pencarian rute baru pada saat pengiriman data akan dilakukan sehingga *delay* menjadi rendah [5].

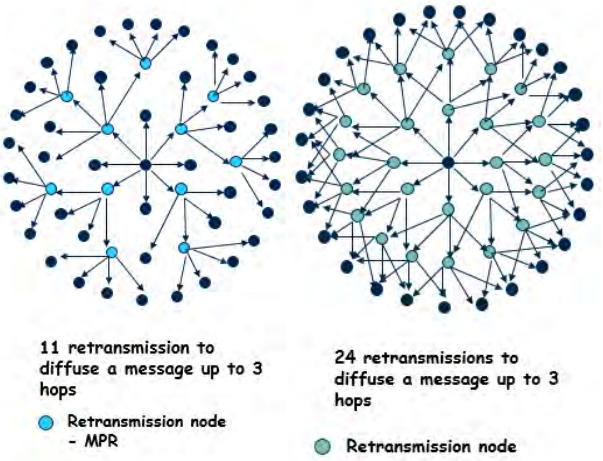
Protokol DSDV digunakan sebagai protokol VANET.

2.2.2. *Optimized Link State Routing* (OLSR)

OLSR merupakan protokol *routing* proaktif, yang dapat dengan segera menyediakan *routing* ke semua *network* tujuan yang ada. Protokol ini merupakan pengembangan dari algoritma *link-state* klasik untuk memenuhi persyaratan dari jaringan nirkabel dinamis seperti MANET. OLSR merupakan protokol *routing* yang menggunakan algoritma *link-state* klasik dan algoritma Djikstra

untuk mencari *shortest path*. Optimalisasi ini berdasarkan pada konsep *Multipoint Relays* (MPR). Setiap *node* menyeleksi *node-node* tetangganya sebagai *Multipoint Relay* (MPR). Pada OLSR, hanya *node* yang berperan sebagai MPR yang bertanggung jawab untuk melanjutkan *control traffic* (paket kontrol), yang dimaksud untuk penyebaran ke seluruh jaringan. *Multipoint Relays* (MPRs) menyediakan mekanisme untuk melakukan *flood control traffic* dengan mengurangi jumlah transmisi yang dibutuhkan.

Pertama dengan menggunakan *multipoint relay* dapat mengurangi ukuran dari *control message*. Daripada menyatakan semua *link*, *node* menyatakan hanya sekumpulan *links* dengan node tetangganya sebagai *"multipoint relay"*. Penggunaan MPR juga meminimalisasi *flooding* dari *control traffic*. Teknik ini secara signifikan mengurangi jumlah transmisi ulang dari *broadcast control* message. Sistem link state mempunyai cara yang lebih efisien dibanding dengan sistem *distance vector*. Jika ada proses pembaharuan, maka hanya pembaharuan itu yang akan dikirimkan. Koleksi jalur terbaik kemudian akan membentuk tabel *routing node*.



Gambar 2-5 Proses packet flooding pada OLSR

Sebuah *node* OLSR yang sedang beroperasi akan secara periodik menyebarkan paket "*hello*" sehingga tetangga dapat mendeteksi keberadaan *node* tersebut. Setiap *node* menghitung berapa jumlah paket "*hello*" yang hilang atau diterima dari tetangga sehingga mendapatkan informasi tentang topologi dan kualitas sambungan *node* lingkungan. Informasi topologi yang diterima di *broadcast* sebagai pesan *topology control* (TC) dan diteruskan oleh tetangga yang dipilih sebagai *multipoint-relay* (MPR). Pada Gambar 2-5 melalui MPR, hanya *node* yang dipilih saja yang bertugas untuk membanjiri jaringan dengan mengirimkan *broadcast-messages* ke seluruh jaringan.

OLSR tepat digunakan untuk jaringan yang besar dan padat karena penggunaan *MPRs*. Semakin besar dan padat suatu jaringan, maka optimasi akan semakin berpengaruh dibanding dengan protokol *link-state* klasik. OLSR menggunakan proses pencarian rute *hop-by-hop* dimana setiap *node* menggunakan informasi *routing table* lokalnya untuk mengirim paket. OLSR lebih efisien apabila bekerja dalam jaringan padat dengan *traffic* yang rendah. OLSR membutuhkan *bandwith* yang konstan untuk memperoleh *topology update message* [6].

Protokol OLSR digunakan sebagai protokol VANET.

2.3. VirtualBox

VirtualBox adalah *software* virtualisasi yang handal untuk mesin berarsitektur x86 dan AMD64 yang dapat digunakan untuk perusahaan maupun untuk rumahan. Tidak hanya penuh dengan fitur, VirtualBox juga merupakan solusi profesional karena produk ini didistribusikan secara gratis dan *opensource* dibawah lisensi *General Public Licence* (GPL).

VirtualBox merupakan salah satu produk perangkat lunak yang sekarang dikembangkan oleh Oracle. Aplikasi ini pertama kali dikembangkan oleh perusahaan Jerman, Innotek GmbH. Pada Februari 2008, Innotek GmbH diakusisi oleh Sun Microsystems. Sun Microsystem kemudian juga diakuisisi oleh Oracle.

Oracle VM VirtualBox adalah perangkat lunak virtualisasi, yang dapat digunakan untuk mengeksekusi sistem operasi "tambahan" di dalam sistem operasi "utama". Sebagai contoh, jika seseorang mempunyai sistem operasi Windows yang terpasang di komputernya, maka yang bersangkutan dapat pula menjalankan sistem operasi lain yang diinginkan di dalam sistem operasi MS Windows tersebut. Fungsi ini sangat penting jika seseorang ingin melakukan uji coba dan simulasi instalasi suatu sistem tanpa harus kehilangan sistem yang ada [7].

Pada tugas akhir ini VirtualBox yang digunakan adalah versi 4.3.28. VirtualBox digunakan untuk menjalankan sistem operasi Linux yang berisi NS-3 dan sebagai penghubung antara sistem operasi Windows dan Linux.

2.4. Simulation of Urban Mobility (SUMO)

Simulation of Urban Mobility atau dikenal dengan SUMO adalah aplikasi *traffic simulator* yang bersifat *open source* dan *portable* dan dapat menangani jaringan yang sangat besar. SUMO menggunakan model dengan mobilitas mikroskopis dan berkesinambungan. SUMO dikembangkan oleh Daniel Krajzewicz, Eric Nikolay, dan Michael Behrisch di *Institute of Transportation Systems* (ITS) yang bertempat di *German Aerospace Center,* Berlin pada tahun 2000 yang bertujuan untuk mengakomodasi penelitian-penelitian yang melibatkan pergerakan kendaraan di jalan raya, terutama daerah-daerah yang padat penduduknya [8].

SUMO terdiri dari banyak *tools* yang memiliki fungsi berbeda-beda. Berikut merupakan penjelasan dari fungsi *tools* yang digunakan pada pembuatan Tugas Akhir ini:

- netgenerate.exe
  Netgenerate merupakan *tools* yang berfungsi untuk membuat peta seperti *grid*, *spider* dan bahkan *random network*. Netgenerate juga berfungsi untuk menentukan kecepatan maksimum jalan dan membuat *traffic light*

pada peta. Hasil dari netgenerate ini berupa *file* dengan ekstensi .net.xml.

- netconvert.exe
  Netconvert merupakan *tools* yang berfungsi untuk mengkonversi peta hasil *capture* dari OpenStreetMap yang berekstensi .osm agar sesuai dengan format dari SUMO yaitu .net.xml.

- randomTrips.py
  randomTrips.py merupakan *tools* pada SUMO untuk membuat *node* beserta titik awal dan tujuannya secara acak. Hasil dari randomTrips.py adalah berupa *file* dengan ekstensi .trips.xml.

- duarouter.exe
  Duarouter berfungsi untuk membuat rute pergerakan yang sudah asal dan tujuannya dibuat oleh randomTrips.py. Secara *default* duarouter menggunakan algoritma Djikstra untuk membuat rutenya. Hasil dari duarouter berupa file dengan ekstensi .rou.xml

- sumo-gui.exe
  Sumo-gui berfungsi untuk menampilkan pergerakan setiap *node* yang sudah dibuat sebelumnya secara grafikal. Cara kerja sumo-gui adalah dengan menggabungkan peta yang ada pada *file* berekstensi .net.xml dengan rutenya yang berekstensi .rou.xml dalam sebuah *file* lain dengan format .sumocfg.

- sumo.exe
  sumo.exe berfungsi untuk membuat *file* berekstensi .xml yang kemudian dapat diekspor ke dalam berbagai bahasa simulasi salah satunya NS-3 dengan tools traceExporter.py

- traceExporter.py
  TraceExporter.py digunakan untuk mengekspor hasil *file* SUMO yang berekstensi .xml agar bisa digunakan pada NS-3 dengan menggunakan fungsi NS2mobility-output pada traceExporter.

Pada Tugas Akhir ini SUMO yang digunakan adalah SUMO versi 0.22.0. SUMO digunakan untuk membuat skenario uji coba dalam peta grid dan riil.

2.5. OpenStreetMap

OpenStreetMap adalah proyek untuk membangun sebuah *database* geografis yang gratis untuk dunia. Data tersebut digunakan untuk membangun peta dunia dan peta-peta khusus yang diturunkan dan dimanfaatkan untuk beragam kebutuhan termasuk navigasi. OpenStreetMap memungkinkan siapa saja untuk melihat, mengedit dan menggunakan data geografis yang telah dibangun secara kolaboratif dari mana dan oleh siapa saja di permukaan bumi.

Inti dari proyek ini adalah berupa sebuah *database* geografis mirip Wiki yang dapat dimanfaatkan secara bebas berdasarkan Lisensi *Open Database*. Dengan demikian, pemanfaatan untuk keperluan untuk di cetak, *website* dan untuk aplikasi perangkat lunak seperti navigasi tidak dibatasi maupun ditarik biaya, asalkan menyebutkan OpenStreetMap sebagai sumber data [9]. OpenStreetMap tidak lain merupakan Wikipedia untuk data pemetaan dunia yang dapat diakses melalui *website openstreetmap.org*.

Pada Tugas Akhir ini OpenStreetMap digunakan sebagai penyedia peta untuk skenario Surabaya.

2.6. JOSM

Java OpenStreetMap Editor (JOSM) adalah sebuah aplikasi *desktop* yang dibuat menggunakan teknologi Java dan pengoperasiannya dapat berjalan pada sistem operasi Windows, Mac OS, dan Linux. JOSM digunakan sebagai salah satu *editor*

data geospasial dari OSM yang berfungsi untuk melakukan digitasi pada data spasial OSM. *Website* JOSM dapat diakses di *josm.openstreetmap.de* untuk dapat mengunduh versi terakhir dari aplikasi ini.

JOSM memiliki banyak fitur *built-in*, seperti dukungan pemetaan audio dan foto, yang membantu mengubah informasi survei menjadi peta. JOSM ini juga mendukung sistem *plugin* yang dapat menambahkan beberapa fungsi tambahan, seperti digitasi langsung dari *file* GPS log secara *real time*, dan alat-alat menggambar titik, garis, relasi dan lainnya. Ada 3 mode operasi utama yang digunakan ketika pengeditan di JOSM, diantaranya:

- *Select* : untuk memilih elemen atau objek, edit atau melihat *tag* atau attribut.
- *Add*: untuk menambahkan titik baru yang bertujuan membuat jalan/fasilitas umum baru, dan memperpanjang jalan yang ada.
- *Delete*: untuk menghapus elemen atau objek [10].

Pada tugas akhir ini digunakan JOSM versi 8339 untuk melakukan *editing* pada peta riil hasil *capture* dari OpenStreetMap.

2.7. Awk

Awk adalah bahasa pemrograman yang biasanya digunakan sebagai ekstraksi data dan alat pelaporan. Ini adalah fitur standar yang paling mirip sistem operasi Unix. Awk diciptakan di Bell Labs pada Tahun 1970, dan namanya berasal dari nama keluarga penulisnya – Alfred V. Aho, Peter J. Weinberger, dan Brian W. Kernighan. Awk atau juga disebut Gawk (GNU awk), yaitu bahasa pemrograman umum dan utility standard POSIX 1003.2. Jika kecepatan merupakan hal yang penting, Awk adalah bahasa yang sangat sesuai [11].

Fungsi dasar awk adalah untuk mencari *file* per baris (atau unit teks lain) yang berisi pola tertentu. Ketika suatu baris sesuai dengan pola, awk melakukan aksi yang khusus pada baris tersebut. Awk tetap memproses baris input sedemikian hingga mencapai akhir baris input.

Program pada awk berbeda dari program di kebanyakan bahasa lain, karena program awk bersifat "data-driven"; yang mana butuh untuk mendeskripsikan data yang dikehendaki untuk bekerja dan kemudian apa yang akan dilakukan saat data tersebut ditemukan. Kebanyakan bahasa lainnya bersifat "procedural"; maka dari itu butuh mendeskripsikannya secara detail setiap langkah program yang harus dijalankan. Ketika bekerja dengan bahasa prosedural, biasanya sangat sulit untuk mendeskripsikan data yang hendak diproses oleh program. Oleh karena itu, program awk sering kali terasa lebih mudah untuk ditulis dan dibaca.

Pada tugas akhir ini AWK digunakan untuk membuat *script* untuk menghitung *Packet Delivery Ratio* (PDR), *End-to-End Delay* (E2D), dan *Routing Overhead* (RO) dari hasil trace NS-3.

2.8. Network Simulator 3 (NS-3)

Simulator NS-3 adalah sebuah *network simulator* yang memiliki ciri tersendiri yang ditargetkan secara utama untuk tujuan riset dan pendidikan. Proyek NS-3, dimulai pada tahun 2006, adalah sebuah proyek *open source* yang diatur oleh komunitas peneliti dan pengembang. NS-3 utamanya digunakan pada sistem Linux, namun dapat digunakan untuk FreeBSD, Cygwin (Untuk Windows), dan dukungan untuk Visual Studio yang bersifat *native* yang masih dikembangkan. NS-3 mempunyai beberapa fitur yang dapat dimanfaatkan untuk memodelkan dan menguji VANET.

Dengan NS-3, VANET disimulasikan melalui membuat skenario jaringan dan penerapan protokol *routing*. Pada modul NS-3 versi 3.22 terdapat *source* untuk melakukan percobaan lingkungan VANET. Pembuatan topologi, *node* dan protokol yang digunakan untuk VANET sudah didukung oleh NS-3. Skenario *mobility* dengan format NS-2 juga dapat digunakan di NS-3 dengan bantuan *library* atau *helper* yang telah tersedia. Dengan NS-3 kita dapat menambahkan fungsi-fungsi baru di dalam *core* NS-3 karena NS-3 bersifat *open source*. NS-3 dikembangkan menggunakan bahasa C++ di lapisan inti dan *script python*. Fitur-fitur NS-3 di antaranya adalah sistem atribut NS-3 terdokumentasi dengan baik.

Setiap objek NS-3 memiliki seperangkat atribut (*name*, *type*, *initial value*) dan NS-3 selaras dengan sistem nyata.

NS-3 menyediakan berbagai banyak modul yang mendukung berbagai macam simulasi skenario. Simulasi yang efektif untuk *vehicular ad hoc network* (VANET) membutuhkan model yang bisa menangkap jaringan *wireless* yang sangat dinamis, propagasi dan karakter mobilitas dari skenario. Representasi hasil data simulasi pada NS-3 dapat ditampilkan dalam bentuk grafik, sehingga memudahkan untuk menganalisis dan mengevaluasi hasil terhadap suatu model jaringan VANET [12].

Pada Tugas Akhir ini digunakan NS-3 versi 3.22 sebagai aplikasi simulasi skenario VANET dengan protokol routing DSDV dan OLSR.

2.8.1. Instalasi NS-3

Dokumentasi tentang cara instalasi NS-3 bersumber dari Wiki nsnam.org. terdapat banyak pilihan cara instalasi, tapi pada Tugas Akhir ini digunakan cara melakukan *download* repositori pada server manual menggunakan Tarball dan membangun NS-3 dengan *script* build.py. Langkah-langkah secara detail sebagai berikut.

- Pertama kali dilakukan yaitu instalasi modul gcc, g++, python, python-dev, bzr, mercurial, gdb, valgrind, gsl-bin, libgsl0-dev, libgsl0ldbl, flex, bison, libfl-dev, tcpdump, sqlite, sqlite3, libsqlite3-dev, libxml2, libxml2-dev, libgtk2.0-0, libgtk2.0-dev, vtun, lxc, uncrustify, doxygen, graphviz, imagemagick, texlive, texlive-extra-utils, texlive-latex-extra, python-sphinx, dia, python-pygraphviz, python-kiwi, python-pygoocanvas, libgoocanvas-dev, libboost-signals-dev, dan libboost-filesystem-dev yang menjadi dependensi untuk instalasi NS-3.

```
$ sudo apt-get install gcc g++ python python-dev
bzr mercurial gdb valgrind gsl-bin libgsl0-dev
libgsl0ldbl flex bison libfl-dev tcpdump sqlite
sqlite3 libsqlite3-dev libxml2 libxml2-dev
libgtk2.0-0 libgtk2.0-dev vtun lxc uncrustify
doxygen graphviz imagemagick texlive texlive-extra-
utils texlive-latex-extra python-sphinx dia python-
pygraphviz python-kiwi python-pygoocanvas
libgoocanvas-dev libboost-signals-dev libboost-
filesystem-dev
```

Gambar 2-6 Perintah untuk instalasi dependensi NS-3

- Setelah semua dependensi lengkap, dilakukan unduh modul ns-3 menggunakan Tarball dan ekstraksi *file* modul dengan perintah berikut.

```
$ cd
$ mkdir workspace
$ cd workspace
$ wget https://www.nsnam.org/releases/ns-allinone-
3.22.tar.bz2
$ tar xjf ns-allinone-3.20.tar.bz2
```

Gambar 2-7 Perintah untuk mengunduh NS-3

- *File* Tarbal NS-3 yang telah selesai diunduh kemudian dilakukan proses *build* dengan program build.py yang ada pada direktori ns-allinone-3.22. Program ini akan mengkonfigurasi NS-3 sesuai dengan keperluan pengguna pada umumnya. Oleh karena pada Tugas Akhir ini membutuhkan modul example dan test, maka dari itu disisipkan pada argumen untuk build.py. Perintahnya dan hasil running seperti berikut.

```
$ ./build.py --enable-examples --enable-tests
```

Gambar 2-8 Perintah untuk build NS-3

Output setelah melakukan build menggunakan build.py yaitu daftar modul yang berhasil dibangun dan yang yang tidak dibangun pada NS-3. Output build.py dapat dilihat pada gambar berikut.

```
Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout
dsdv                     dsr                      energy
fd-net-device            flow-monitor             internet
lr-wpan                  lte                      mesh
mobility                 mpi                      netanim (no Python)
network                  nix-vector-routing       olsr
point-to-point           point-to-point-layout    propagation
sixlowpan                spectrum                 stats
tap-bridge               test (no Python)         topology-read
uan                      virtual-net-device       visualizer
wave                     wifi                     wimax

Modules not built (see ns-3 tutorial for explanation):
brite                    click                    openflow
```

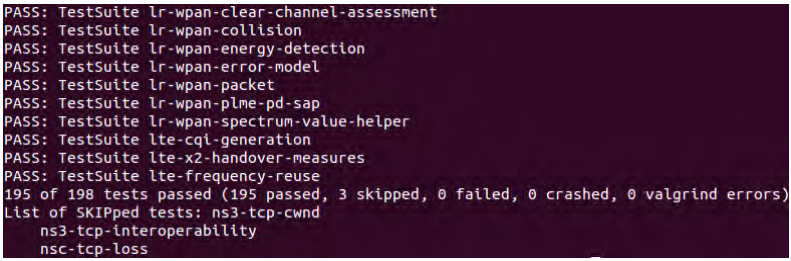
Gambar 2-9 Output build.py pada NS-3

- Untuk melakukan tes fungsi-fungsi pada ns-3, dilakukan perintah berikut.

```
$ ./test.py -c core$ ./test.py -c core
```

Gambar 2-10 Perintah untuk malukan tes fungsi-fungsi pada NS-3

Tes tersebut berjalan secara paralel oleh Waf dan hasilnya memunculkan status per tes yang dilakukan dan pada bagian akhir terdapat *report* seperti berikut.



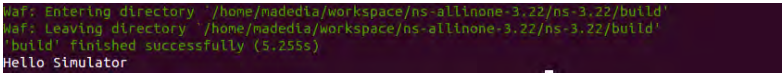
Gambar 2-11 Output test.py pada NS-3

- Terdapat 3 tes yang dilewatkan atau berstatus SKIPPED namun tidak terjadi masalah. Untuk percobaan *running* salah satu program NS-3, dapat dilakukan menjalankan contoh program hello-simulator dengan perintah berikut.

```
./waf --run hello-simulator
```

Gambar 2-12 Perintah menjalankan program hello-simulator

Hasil dari program hello-simulator dapat dilihat seperti berikut.



Gambar 2-13 Hasil output program hello-simulator

2.8.2. Penggunaan vanet-routing-compare.cc

Pada ns-3, terdapat *script* program untuk VANET yaitu vanet-routing-compare.cc yang terletak pada direktori src/wave/examples Program ini mengkombinasikan beberapa modul ns-3 dari examples untuk mendukung pengembangan simulasi VANET.

Script vanet-routing-compare memiliki berbagai jenis parameter simulasi yang dapat dikonfigurasi sesuai kebutuhan. Kategori parameter yang dapat dirubah nilai aslinya dan

kegunaannya serta contoh perintah menjalankan *script* dengan konfigurasi yang dilakukan ditunjukkan pada tabel berikut.

Tabel 2-1 Parameter pada vanet-routing-compare.cc

Parameter	Meaning	Values	Default	Example
totaltime	Simulation end time	Double	300.01	./waf --run "scratch/my-vanet-routing-compare --totaltime=200"
nodes	Number of nodes (i.e. vehicles)	integer	156	./waf --run "scratch/my-vanet-routing-compare --nodes=50"
sinks	Number of routing sinks	integer	10	./waf --run "scratch/my-vanet-routing-compare --sinks=1"
txp	Transmit power (dB), e.g. txp=7.5	Double	7.5	./waf --run "scratch/my-vanet-routing-compare --txp=7"
protocol	1=OLSR;2=AODV;3=DSDV;4=DSR	integer	2	./waf --run "scratch/my-vanet-routing-

Parameter	Meaning	Values	Default	Example
				compare --protocol=2"
lossModel	1=Friis;2=ItuR1411Los;3=TwoRayGround;4=LogDistance	integer	3	./waf --run "scratch/my-vanet-routing-compare --lossModel=3"
fading	0=None;1=Nakagami;(buildings=1 overrides)	integer	0	./waf --run "scratch/my-vanet-routing-compare --fading=1"
phyMode	Wifi Phy mode	String	OfdmRate6MbpsBW10MHz	./waf --run "scratch/my-vanet-routing-compare --phyMode=OfdmRate24MbpsBW10MHz"
80211Mode	1=802.11p; 2=802.11b; 3=WAVE-PHY	integer	1	./waf --run "scratch/my-vanet-routing-compare --80211Mode=1"
traceFile	Ns2 movement trace file	String	./src/wave/examples/low	./waf --run "scratch/my-vanet-

Parameter	Meaning	Values	Default	Example
			_ct-unterstrass-1day.filt.5.adj.mob	routing-compare --traceFile=/scratch/15scene1.tcl"
mobility	1=trace;2=RWP	integer	1	./waf --run "scratch/my-vanet-routing-compare --mobility=1"
rate	Rate	String	512bps	./waf --run "scratch/my-vanet-routing-compare --rate=2048bps"
phyModeB	Phy mode 802.11p	String	DsssRate11Mbps	./waf --run "scratch/my-vanet-routing-compare --phyModeB=DsssRate11Mbps"
speed	Node speed (m/s)	integer	20	./waf --run "scratch/my-vanet-routing-compare --speed=20"

Parameter	Meaning	Values	Default	Example
pause	Node pause (s)	integer	0	./waf --run "scratch/my-vanet-routing-compare --pause=0"
verbose	0=quiet;1=verbose	integer	0	./waf --run "scratch/my-vanet-routing-compare --verbose=1"
scenario	1=synthetic, 2=playback-trace	integer	1	./waf --run "scratch/my-vanet-routing-compare --scenario=3"
gpsaccuracy	GPS time accuracy, in ns	Double	40	./waf --run "scratch/my-vanet-routing-compare --gpsaccuracy=40"
txmaxdelay	Tx max delay, in ms	Double	10	./waf --run "scratch/my-vanet-routing-compare --txmaxdelay=10"

Parameter	Meaning	Values	Default	Example
routingTables	Dump routing tables at t=5 seconds	integer	0	./waf --run "scratch/my-vanet-routing-compare --routingTables=1"
asciiTrace	Dump ASCII Trace data	integer	0	./waf --run "scratch/my-vanet-routing-compare --asciiTrace=1"
pcap	Create PCAP files for all nodes	integer	0	./waf --run "scratch/my-vanet-routing-compare --pcap=1"

2.8.3. NS-3 *Trace File*

NS-3 *Trace File* merupakan output hasil *running* dalam bentuk ASCII dari NS3 yang berekstensi .tr. *Trace File* berisi log tentang semua jenis pengiriman dan penerimaan paket yang terjadi selama simulasi. Di dalam *Trace File* tercatat berbagai jenis paket sesuai jenis protokol *routing* yang digunakan. Untuk pengiriman dan penerimaan jenis paket data, format *trace file* tidak berbeda pada protokol DSDV dan OLSR. Letak perbedaannya yaitu pada paket *routing* yang dikirim karena menggunakan *header* masing-masing sesuai protokol DSDV dan OLSR. Tiap baris log ini dianalisis untuk mendapatkan performa protokol. Contoh pengiriman data paket pada NS-3 *Trace File* dapat dilihat pada gambar Gambar 2-14.

```
t 21.7014
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/St
ate/Tx ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:0d,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=12) ns3::LlcSnapHeader
(type 0x800) ns3::Ipv4Header (tos 0x0 DSCP Default
ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes)
0 flags [none] length: 92 10.1.0.2 > 10.1.0.1)
ns3::UdpHeader (length: 72 49153 > 9) Payload
(size=64) ns3::WifiMacTrailer ()
```

Gambar 2-14 Trace pengiriman data paket

Huruf "t" di kolom pertama menandakan pengiriman paket (*transmission*) dan mengandung *header* ns3::Ipv4Header dengan IP asal 10.1.0.2 dari *node* 1 dan IP tujuan 10.1.0.1 dari *node* 0.

Untuk penerimaan data paket seperti berikut tidak jauh beda dengan pengiriman data paket, pembedanya yaitu kolom pertama dengan huruf inisial "r" yang menandakan paket diterima dan format selanjutnya sama persis dengan paket pengiriman.

```
r 22.7256
/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/St
ate/RxOk ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:01,
SA=00:00:00:00:00:07, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=19) ns3::LlcSnapHeader
(type 0x800) ns3::Ipv4Header (tos 0x0 DSCP Default
ECN Not-ECT ttl 57 id 1 protocol 17 offset (bytes)
0 flags [none] length: 92 10.1.0.2 > 10.1.0.1)
ns3::UdpHeader (length: 72 49153 > 9) Payload
(size=64) ns3::WifiMacTrailer ()
```

Gambar 2-15 Trace penerimaan data paket

Contoh format untuk paket *routing* DSDV pada *trace file* seperti Gambar 2-16.

```
t 0.000119
/NodeList/9/DeviceList/0/$ns3::WifiNetDevice/Phy/St
ate/Tx ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=0usDA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:0a, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN
Not-ECT ttl 1 id 0 protocol 17 offset (bytes) 0
flags [none] length: 40 10.1.0.10 > 10.1.255.255)
ns3::UdpHeader (length: 20 269 > 269)
ns3::dsdv::DsdvHeader (DestinationIpv4: 10.1.0.10
Hopcount: 1 SequenceNumber: 2) ns3::WifiMacTrailer
()
```

Gambar 2-16 Trace pengiriman paket routing DSDV

Sementara contoh format untuk paket *routing* OLSR pada *trace file* seperti Gambar 2-17.

```
t 0.00966143
/NodeList/20/DeviceList/0/$ns3::WifiNetDevice/Phy/S
tate/Tx ns3::WifiMacHeader (DATA ToDS=0, FromDS=0,
MoreFrag=0, Retry=0, MoreData=0
Duration/ID=0usDA=ff:ff:ff:ff:ff:ff,
SA=00:00:00:00:00:15, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=0) ns3::LlcSnapHeader (type
0x800) ns3::Ipv4Header (tos 0x0 DSCP Default ECN
Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0
flags [none] length: 48 10.1.0.21 > 10.1.255.255)
ns3::UdpHeader (length: 28 698 > 698)
ns3::olsr::PacketHeader () ns3::olsr::MessageHeader
() ns3::WifiMacTrailer ()
```

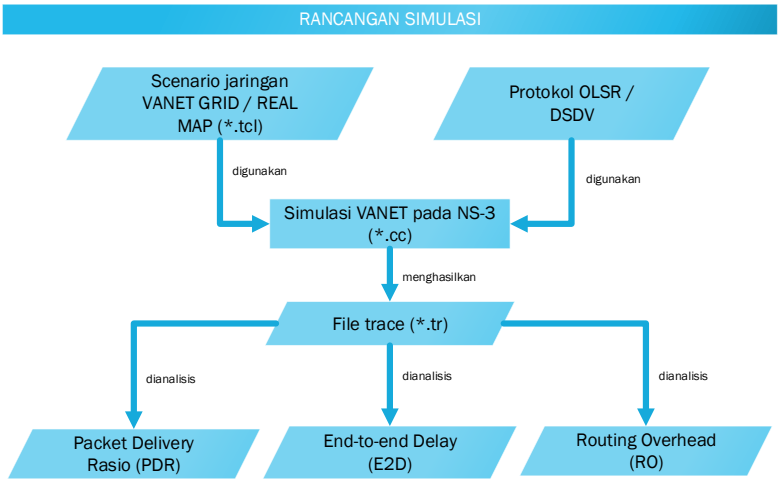
Gambar 2-17 Trace pengiriman paket routing OLSR

BAB III
PERANCANGAN

Perancangan merupakan bagian penting dari pembuatan sistem secara teknis. Sehingga bab ini secara khusus akan menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir. Berawal dari deskripsi umum sistem hingga perancangan skenario, alur dan implementasinya.

3.1. Deskripsi Umum

Pada Tugas Akhir ini akan dilakukan analisis tentang performa protokol dari MANET jenis proaktif yaitu DSDV dan OLSR di lingkungan VANET. Tahapan rancangan simulasi dapat dilihat pada Gambar 3-1.



Gambar 3-1 Tahapan rancangan simulasi

Dalam penelitian ini, terdapat 2 jenis skenario yang digunakan sebagai perbandingan pengukuran performa yaitu peta

berbentuk *grid* dan peta riil lingkungan lalu lintas kota Surabaya. Baik peta grid dan peta riil menggunakan konfigurasi yang sama. Skenario dibuat menggunakan bantuan aplikasi SUMO, peta digital OpenStreetMap, dan aplikasi JOSM. *Tools* tersebut dijalankan pada lingkungan sistem operasi Windows. Sedangkan untuk simulasi, skenario akan dijalankan menggunakan program NS-3 yang bernama vanet-routing-compare pada sistem operasi Linux.

Jumlah kepadatan kendaraan tiap jenis skenario dibuat bervariasi dalam uji coba. Dan hasil proses uji coba dari skenario akan yang dihasilkan dalam bentuk *trace file* dianalisis dengan perhitungan metrik *Packet Delivery Ratio* (PDR), *End-to-End Delay* (E2D), dan *Routing Overhead* (RO). Dari hasil metrik tersebut dianalisis performa protokol dan hal-hal yang mempengaruhi kerja protokol DSDV dan OLSR di lingkungan VANET.

3.2. Perancangan Shared Folder antara Windows dan VirtualBox

Dalam perancangan lintas sistem operasi antara pembuatan skenario yang dirancang di sistem operasi Windows sementara program NS-3 yang dijalankan pada sistem operasi Linux dengan lingkungan VirtualBox, maka pada Tugas Akhir ini dirancang folder *sharing* sebagai jembatan yang memudahkan pemindahan / duplikasi *file*.

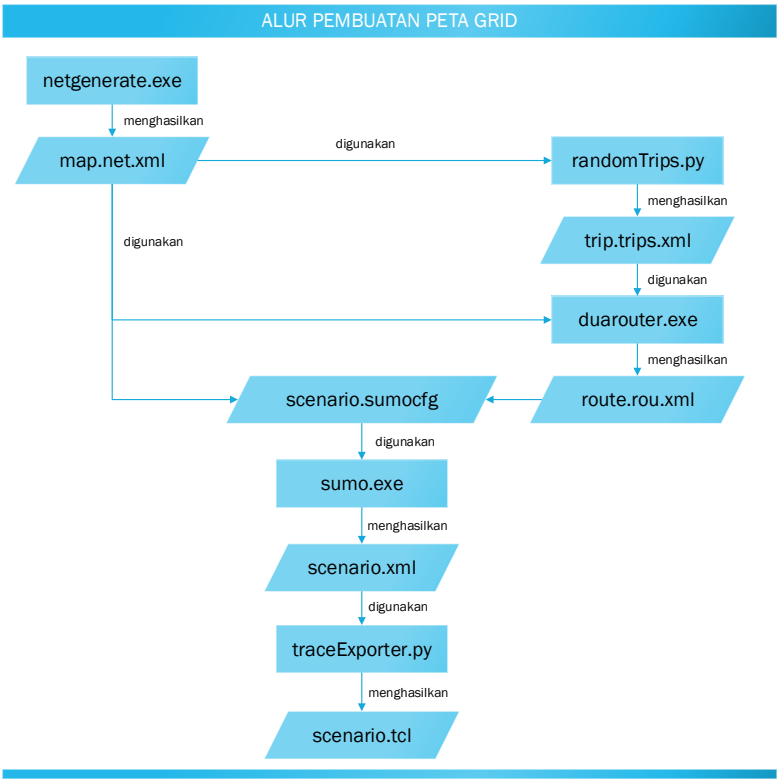
3.3. Perancangan Skenario

Perancangan skenario mobilitas uji coba VANET dimulai dari perancangan peta pergerakan *node*, pembuatan rute lalu lintas kendaraan dan implementasi pergerakan. Dalam Tugas Akhir ini, bagian perencanaan peta pergerakan node menggunakan 2 varian skenario, yaitu peta *grid* dan peta riil. Peta *grid* yang dimaksud berupa jalan-jalan yang saling berpotongan dan membentuk petak yang menggambarkan contoh lingkungan riil dalam bentuk sederhana. Peta *grid* digunakan sebagai tes awal implementasi VANET. Sedangkan untuk peta riil adalah peta yang dapat

menggambarkan lingkungan lalu lintas yang nyata. Peta riil yang digunakan yaitu peta lingkungan lalu lintas kota Surabaya. Peta kota Surabaya merupakan hasil impor dari OpenStreetMap. Pada parameter kecepatan kendaraan maksimal dalam peta diatur dengan kecepatan maksimal sebesar 15 m/s. Untuk penjelasan masing-masing peta adalah sebagai berikut:

3.3.1. Skenario *Grid*

Skenario *grid* dibuat murni menggunakan tools SUMO. Alur pembuatannya dapat dilihat pada Gambar 3-2.



Gambar 3-2 Alur pembuatan skenario grid

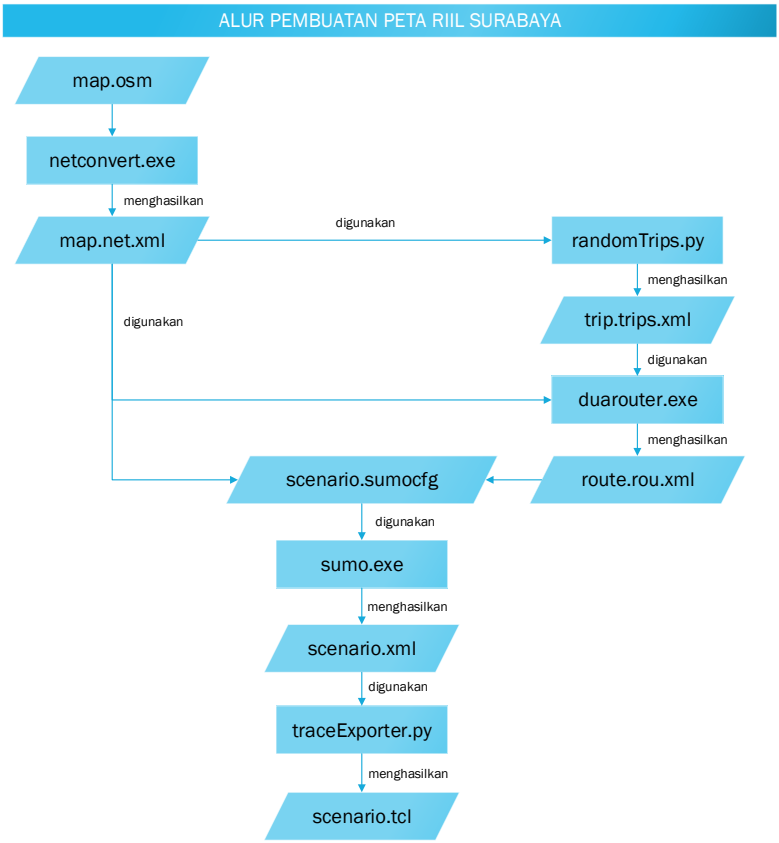
Pembuatan skenario *grid* diawali dengan menentukan panjang dan lebar peta *grid* yang digunakan. Setelah ukuran peta ditentukan, langkah berikutnya adalah menentukan banyak garis horizontal dan garis vertikal. Peta yang digunakan berukuran 1500 m x 1500 m atau 2.25 km^2 dan jumlah garis horizontal dan vertikalnya masing-masing sebanyak 10 (sepuluh). Peta *grid* juga dilengkapi *traffic light* dalam setiap persimpangannya dan batas kecepatannya yaitu sebesar 15 m/s. Semua konfigurasi peta tersebut dijalankan dengan bantuan *tools* netgenerate.exe. Untuk perancangan pergerakan kendaraan pada peta, dilakukan *mapping* secara *random* posisi kendaraan serta rute tujuan kendaraan dengan menggunakan *tools* randomTrips.py serta dilakukan implementasi perutean pergerakan kendaraan sesuai map *grid* dengan menggunakan *tools* duarouter.exe.

Tahap pembuatan skenario grid selanjutnya yaitu menggabungkan peta *grid* dengan rute pergerakan kendaraan dalam *file* simulasi SUMO menggunakan *tools* sumo.exe. *File* simulasi akan berisi kecepatan kendaraan dalam posisi yang dituju mulai dari detik ke-0 simulasi hingga destinasi akhir kendaraan sudah tercapai atau hingga 200 detik simulasi berlangsung. Tahap terakhir ialah konversi skenario SUMO ke dalam format NS-2 *mobility trace* dengan tools traceExporter.py agar dapat dibaca program NS-3 sebagai skenario jaringan VANET.

3.3.2. Skenario Riil

Pembuatan skenario riil Surabaya diawali dengan mencari daerah umum di sekitar kota Surabaya yang mendekati bentuk *grid* dan dengan mobilitas tinggi yang cocok untuk VANET. Karena pembuatan peta agar mendekati dengan aslinya, OpenStreetMap digunakan untuk membantu melakukan *capture* peta. Agar map dapat berjalan baik saat simulasi, hasil impor dari OpenStreetMap disempurnakan menggunakan *tools* JOSM seperti untuk kasus jalan yang terputus saat melakukan *capture*, pengaturan *traffic light signal* yang tidak sesuai pengaturan. Setelah mendekati sempurna, baru dilakukan konversi dari *file* hasil OpenStreetMap

agar dapat dibaca SUMO dengan bantuan *tools* netconvert.exe. Tahap selanjutnya sama dengan pembuatan skenario grid hingga menghasilkan jaringan riil VANET dengan skenario kota Surabaya. Alur pembuatannya dapat dilihat pada Gambar 3-3.



Gambar 3-3 Alur pembuatan skenario riil Surabaya

3.4. Perancangan Simulasi pada NS-3

Pada perancangan kode NS-3 dengan konfigurasi VANET, dilakukan penggabungan skenario mobilitas dengan skrip TCL

serta kode program vanet-routing-compare dari NS-3. Berikut parameter simulasi perancangan sistem VANET yang dijalankan dapat dilihat pada Tabel 3-1.

Tabel 3-1 Parameter simulasi

No.	Parameter	Spesifikasi
1	Network simulator	NS-3, 3.23
2	Routing Protocol	DSDV, OLSR
3	Waktu simulasi	200 detik
4	Area simulasi	1500 m x 1500 m
5	Banyak kendaraan	25, 50, 75, 100
6	Radius transmisi	320 m
7	Kecepatan maksimal	15 m/s
8	Tipe data	Constant Bit Rate (CBR)
9	Source / Destination	Statik (Node 1 / Node 0)
10	Kecepatan generasi paket	1 paket per detik
11	Ukuran paket data	64 bytes / 512 bit per second
12	Protokol MAC	IEEE 802.11p
13	Mode propagasi	Two-ray ground reflection model
14	Tipe mobility	NS-2
15	Mobility model	Peta grid dan peta real Surabaya
16	Tipe kanal	Wireless channel
17	Tipe trace	ASCII Trace

3.5. Perancangan Metrik Analisis

Metrik yang akan dianalisis pada Tugas Akhir ini adalah *Packet Delivery Ratio* (PDR), *End-to-End Delay* (E2D), dan *Routing Overhead* (RO). Penjelasannya sebagai berikut:

3.5.1. *Packet Delivery Ratio* (PDR)

PDR dihitung dari perbandingan antara paket yang dikirim dengan paket yang diterima. PDR dihitung dengan menggunakan persamaan (3-1), dimana *received* adalah banyaknya paket data yang diterima dan *sent* adalah banyaknya paket data yang dikirimkan.

$$PDR = \frac{received}{sent} \times 100\% \quad \text{.....(3-1)}$$

3.5.2. *End-to-End Delay* (E2D)

E2D dihitung dari rata-rata delay antara waktu paket diterima dan waktu paket dikirim. E2D dihitung dengan menggunakan persamaan (3-2), dimana $t_{received[i]}$ adalah waktu penerimaan paket dengan urutan / id ke-i, $t_{sent[i]}$ adalah waktu pengiriman paket dengan urutan / id ke-i, dan sent adalah banyaknya paket data yang dikirimkan.

$$E2D = \frac{\sum_{i \leq sent}^{i=0} t_{received[i]} - t_{sent[i]}}{sent} \quad \text{.....(3-2)}$$

3.5.3. *Routing Overhead* (RO)

RO adalah jumlah paket kontrol *routing* yang ditransimisikan per data paket yang terkirim ke tujuan selama simulasi terjadi. RO dihitung berdasarkan paket *routing* yang ditransmisikan baik yang melalui 1 *hop* atau *multiple hop*. Baris yang mengandung *routing overhead* pada *trace file* ditandai dengan paket yang bertipe transmisi (t) dan terdapat *header* paket dari protokol yang digunakan.

BAB IV
IMPLEMENTASI

Bab ini merupakan bahasan mengenai implementasi dari perancangan sistem yang telah dijabarkan pada bab sebelumnya.

4.1. Lingkungan Pembangunan Perangkat Lunak

Pembangunan perangkat lunak dilakukan pada lingkungan pengembangan sebagai berikut:

4.1.1. Lingkungan Perangkat Lunak

Adapun perangkat lunak yang digunakan untuk pengembangan sistem adalah sebagai berikut:

- Sistem Operasi berupa Windows 8.1 Pro 64 bit untuk lingkungan SUMO dan Sistem Operasi berupa Ubuntu 14.04 LTS 64 bit untuk lingkungan NS-3,
- Virtual Box untuk pengembangan aplikasi NS-3 dan analisis,
- SUMO versi 0.23.0 untuk mendesain skenario mobilitas VANET,
- Google Chrome versi 43.0.2357.81 m (64-bit) untuk mengoperasikan web OpenSteetMap, dan
- JOSM versi 8339 sebagai editor peta hasil ekspor dari OpenStreetMap.

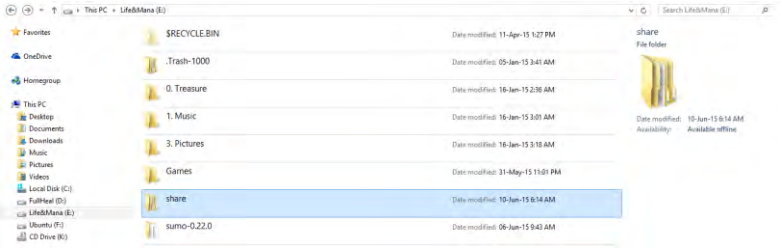
4.1.2. Lingkungan Perangkat Keras

Spesifikasi perangkat keras yang digunakan sebagai lingkungan implementasi perangkat lunak Tugas Akhir adalah sebagai berikut:

- P*rocessor* Intel(R) Core(TM) i7-2630QM CPU @ 2.0GHz,
- RAM sebesar 8 GB DDR3, dan
- Media penyimpanan sebesar 750 GB.

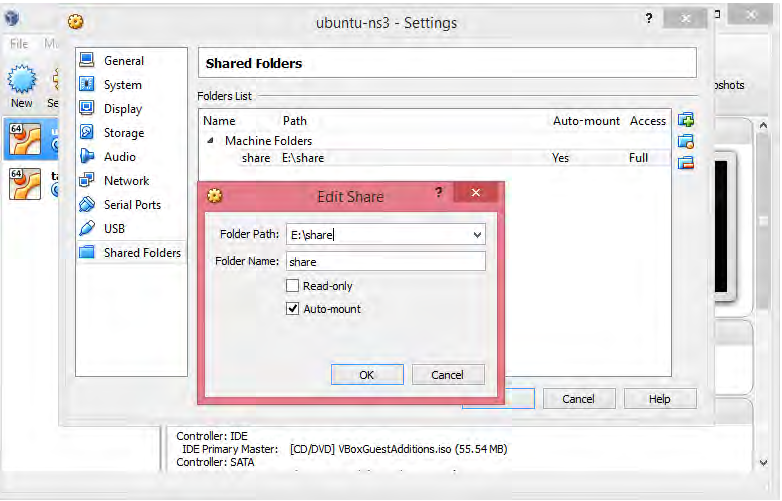
4.2. Implementasi Shared Folder antara Windows dan VirtualBox

Implementasi pembuatan *Shared Folder* antara Windows dan VirtualBox dimulai dengan membuat folder bernama “share “ pada salah satu drive pada Windows misalnya drive E:\ seperti yang ditunjukkan pada Gambar 4-1.



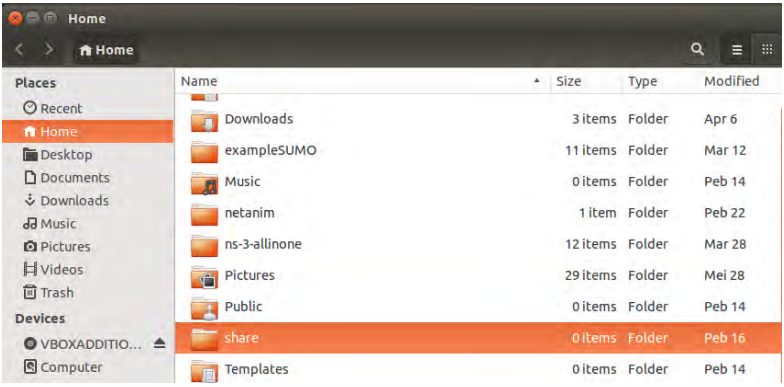
Gambar 4-1 Pembuatan folder share pada Windows

Sementara pada VirtualBox *Machine* dilakukan konfigurasi penambahan *shared folder* dan dicentang pada bagian *Auto Mount* seperti yang ditunjukkan pada Gambar 4-2.



Gambar 4-2 Konfigurasi Shared Folders pada VirtualBox

Kemudian dibuat folder “share” pada Linux pada direktori *home* seperti Gambar 4-3 dan langkah selanjutnya adalah melakukan *mount* pada *shared folder* dengan perintah seperti Gambar 4-4.



Gambar 4-3 Pembuatan folder share pada Ubuntu

```
$ sudo mount -t vboxsf share ~/share/
```

Gambar 4-4 Perintah mount shared folder

4.3. Implementasi Skenario

Implementasi skenario mobilitas VANET dibagi menjadi 2 bagian, yaitu implementasi skenario *grid* dan skenario riil kota Surabaya.

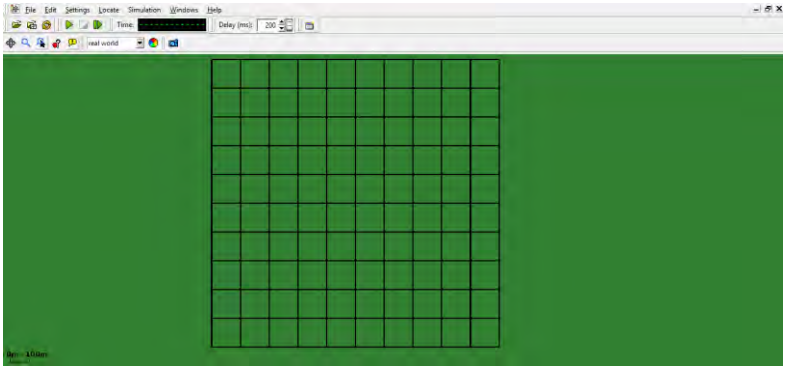
4.3.1. Skenario *Grid*

Dalam implementasi skenario *grid*, melalui *tools* netgenerate dari aplikasi SUMO ditentukan luas peta yaitu 1500 m x 1500 m dengan jumlah petak dari atas ke bawah sebanyak 10 petak dan dari kiri ke kanan sebanyak 10 petak. Sehingga diperlukan jumlah titik persimpangan antara jalan vertikal dan horizontal sebanyak 11 titik x 11 titik dan luasan per petak 150 m x 150 m. Untuk kecepatan kendaraan di peta *grid* yang diperbolehkan juga diatur yaitu sebesar 15 m/s. Semua konfigurasi di atas dijalankan melalui

jendela Command Prompt di Windows dengan direktori di folder SUMO berada seperti Gambar 4-5. Hasil dari pembentukan peta *grid* dapat dilihat pada sumo-gui seperti Gambar 4-6.

```
bin\netgenerate.exe -grid --grid.number=11 --
grid.length=150 --default.speed=15 --tls.guess=1 --
output-file=map.net.xml
```

Gambar 4-5 Perintah netgenerate peta grid



Gambar 4-6 Hasil generate peta grid

Setelah peta terbentuk, maka dilakukan implementasi pergerakan *node* / kendaraan dalam peta menggunakan modul randomTrips.py seperti Gambar 4-7.

```
tools\trip\randomTrips.py -n map.net.xml -e 100 -l
--trip-attributes="departLane=\"best\"
departSpeed=\"max\" departPos=\"random_free\"" -o
trip.trips.xml
```

Gambar 4-7 Perintah randomTrips.py

File trip.trips.xml kemudian diterjemahkan dalam bentuk rute dengan *tools* duarouter.exe seperti Gambar 4-8.

```
bin\duarouter.exe –n speed15.net.xml -t
trips1.trips.xml -o route.rou.xml --ignore-errors –
repair
```

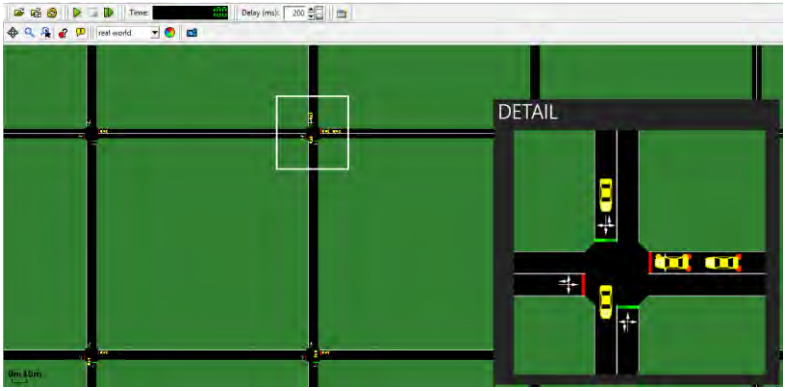
Gambar 4-8 Perintah duarouter.exe

Tools sumo.exe digunakan untuk menggabungkan *file* peta dan rute pergerakan menjadi sebuah skenario dalam bentuk *file* xml dengan bantuan berkas skrip scenario.sumocfg. *File* sumocfg yang dibuat seperti Gambar 4-9.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/x
sd/sumoConfiguration.xsd">

    <input>
        <net-file value="map.net.xml"/>
        <route-files value="route.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="200"/>
    </time>

</configuration>
```

Gambar 4-9 Script file scenario.sumocfg skenario grid

File sumocfg disimpan pada direktori yang sama dengan map.net.xml dan route.rou.xml. Untuk percobaan sebelum dikonversi, berkas sumocfg dibuka melalui aplikasi SUMO. Pada gambar Gambar 4-10 adalah cuplikan pergerakan kendaraan pada beberapa persimpangan jalan.



Gambar 4-10 Cuplikan pergerakan kendaraan pata skenario grid

Untuk dikonversi menjadi sebuah *file* xml yang berisi peta *grid* dan pergerakan kendaraan dijalankan perintah pada Gambar 4-11.

```
bin\sumo.exe -c scenario.sumocfg --fcd-output
scenario.xml
```

Gambar 4-11 Perintah sumo.exe

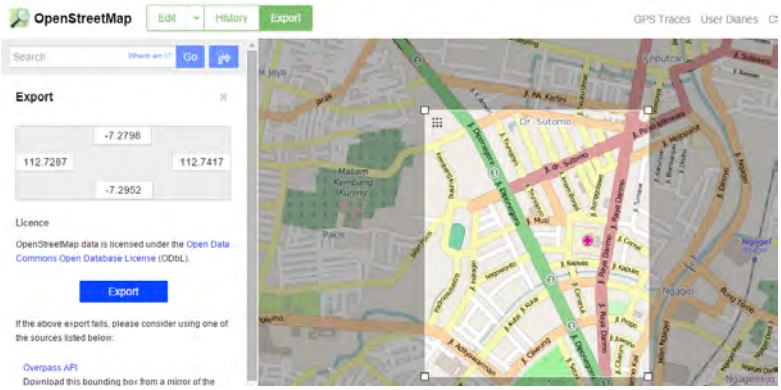
Berkas scenario.xml kemudian dikonversi menjadi *file* pergerakan dalam format mobilitas NS-2 dengan bantuan modul traceExporter.py seperti pada Gambar 4-12.

```
tools\traceExporter.py --fcd-input=scenario.xml --
ns2mobility-output=scenario.tcl
```

Gambar 4-12 Perintah traceExporter.py

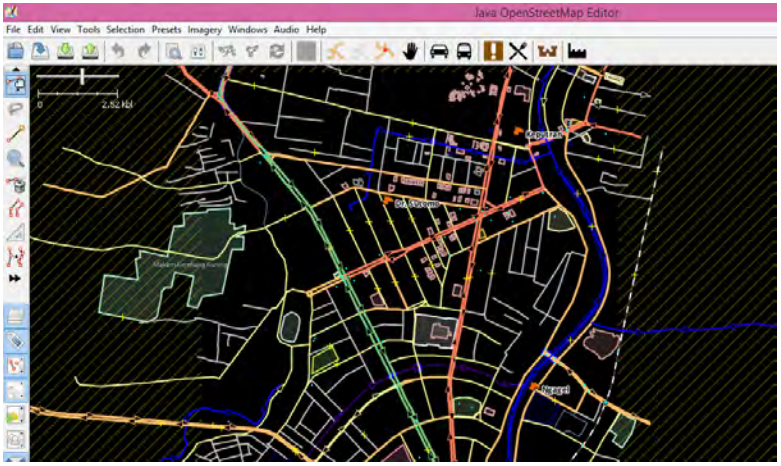
4.3.2. Skenario Riil

Skenario riil menggunakan peta Surabaya yang diambil dari OpenStreetMap dengan cara menandai wilayah kemudian yang digunakan kemudian diekspor dan secara otomatis akan dilakukan *download* oleh program melalui *browser* yang menghasilkan *file* berekstensi .osm seperti pada Gambar 4-13.



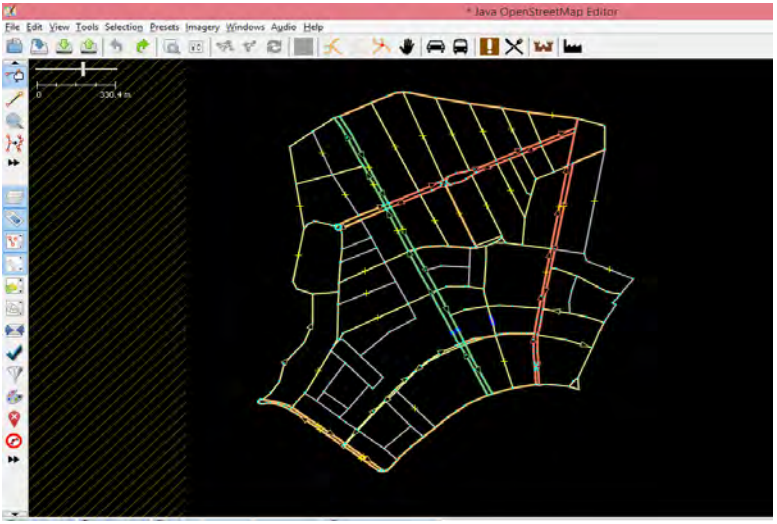
Gambar 4-13 Ekspor peta OpenStreetMap

Peta yang sudah diambil dari OpenStreetMap kemudian dilakukan *editing* menggunakan JOSM untuk mengisikan lampu merah pada tiap persimpangan jalan besar. Jalan yang tidak digunakan dihapus serta bentuk dan ukuran peta dibuat supaya hampir menyerupai peta grid. Kecepatan pada peta riil Surabaya disamakan dengan peta grid yaitu 15 m/s. Gambar 4-14 merupakan *screenshot* saat melakukan *editing* pada JOSM.



Gambar 4-14 Hasil ekspor peta Surabaya pada JOSM

Hasil setelah melakukan *editing* dapat dilihat pada Gambar 4 9. Antar jalan dihubungkan agar tidak terjadi rute buntu. Selain itu juga traffic light juga diisikan di setiap persimpangan.



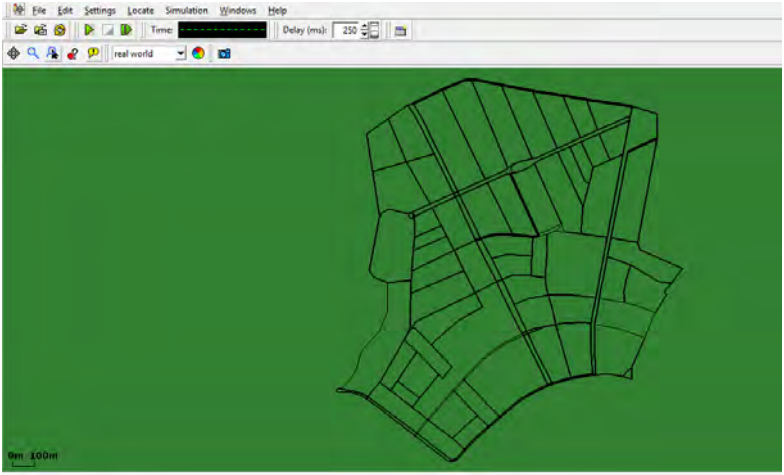
Gambar 4-15 Hasil editing peta Surabaya pada JOSM

Selesai *editing file* osm dikonversi dalam bentuk net.xml agar bisa dibaca oleh program SUMO menggunakan *tools* netconvert. Perintahnya sebagai berikut.

```
bin\netconvert.exe --osm-files map.osm --
default.speed=15 --output-file map.net.xml
```

Gambar 4-16 Perintah netconvert.exe peta Surabaya

Hasil konversi dari *file* osm OpenStreetMap dapat dibuka melalui program sumo-gui terlihat seperti Gambar 4-17. Bentuk peta dari OpenStreetMap yang telah mengalami editing sama persis dengan peta yang telah dikonversi ke SUMO lengkap dengan simulasi traffic light.



Gambar 4-17 Hasil konversi peta Surabaya pada sumo-gui

Setelah peta terbentuk, maka dilakukan implementasi pergerakan *node* atau kendaraan dalam peta menggunakan modul randomTrips.py seperti pada Gambar 4-18.

```
tools\trip\randomTrips.py -n map.net.xml -e 100 -l
--trip-attributes="departLane=\"best\"
departSpeed=\"max\" departPos=\"random_free\"" -o
trip.trips.xml
```

Gambar 4-18 Perintah randomTrips.py

Berkas trip.trips.xml kemudian diterjemahkan dalam bentuk rute dengan *tools* duarouter.exe

```
bin\duarouter.exe -n map.net.xml -t trip.trips.xml
-o route.rou.xml --ignore-errors -repair
```

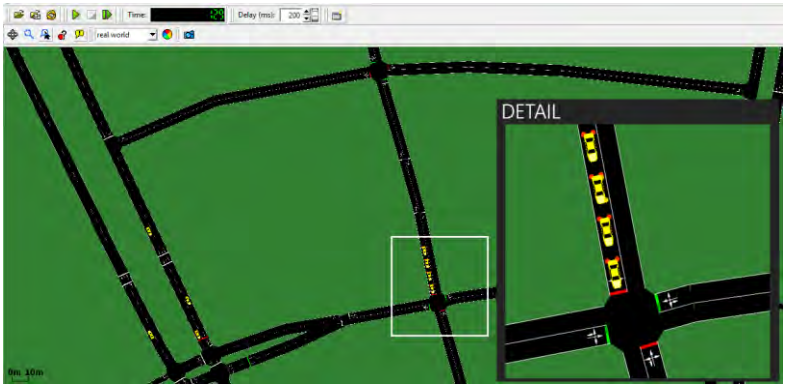
Gambar 4-19 Perintah duarouter.exe

Berkas tipe sumocfg untuk skenario riil kota Surabaya seperti pada Gambar 4-20.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/x
sd/sumoConfiguration.xsd">

    <input>
        <net-file value="map.net.xml"/>
        <route-files value="route.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <end value="200"/>
    </time>

</configuration>
```

Gambar 4-20 Script file scenario.sumocfg skenario Surabaya

Hasil percobaan *running* dalam SUMO seperti Gambar 4-21.



Gambar 4-21 Cuplikan pergerakan skenario Surabaya

Berkas sumocfg disimpan pada direktori yang sama dengan map.net.xml dan route.rou.xml dan dijalankan perintah seperti pada Gambar 4-22.

```
bin\sumo.exe -c scenario.sumocfg --fcd-output
surabaya.xml
```

Gambar 4-22 Perintah sumo.exe

File scenario.xml kemudian dikonversi menjadi *file* pergerakan dalam format mobilitas NS-2 dengan bantuan modul traceExporter.py seperti pada Gambar 4-23.

```
tools\traceExporter.py --fcd-input=surabaya.xml --
ns2mobility-output=surabaya.tcl
```

Gambar 4-23 Perintah traceExporter.py

4.4. Implementasi Simulasi pada NS-3

Untuk mensimulasikan VANET dalam lingkungan NS-3, dilakukan konfigurasi pada program vanet-routing-compare.cc untuk pembangkitan skenario *grid* maupun Surabaya untuk kedua protokol *routing* DSDV dan OLSR. Parameter simulasi vanet-routing-compare dikonfigurasi sesuai dengan kebutuhan skenario dimana *mobility* yang digunakan adalah mobiliy yang berformat NS-2. Parameter yang dikonfigurasi nilainya yaitu:

- ASCII *trace* dan *trace_mobililty* diaktifkan (set 1),
- *mobility* bertipe *trace* dan *trace file* yang digunakan dipanggil ke program,
- protokol dengan nilai 3 adalah protokol DSDV, sedangkan protokol bernilai 1 adalah protokol OLSR,
- jumlah *node* disesuaikan dengan variasi,
- *transmit power* di set 7 dB,
- *node sink* di set 1 (*node* 1 sebagai *sender*, *node* 0 sebagai *receiver*),
- dan *simulation time* di set 200 detik.

Berikut cuplikan kode konfigurasi untuk pemanggilan skenario *grid* pada VANET dengan protokol OLSR dan jumlah *node* sebanyak 50 seperti pada Gambar 4-24.

```
if (m_scenario == 1){
      m_asciiTrace = 1;
      m_protocol = 1;
      m_traceMobility = 1;
      m_mobility = 1;
      m_traceFile =
"scenario/surabaya/sby_scen10.tcl";
      m_nNodes = 50;
      m_txp = 7;
      m_nSinks = 1;
      m_TotalSimTime = 200; }
```

Gambar 4-24 Contoh script skenario vanet-routing-compare

Nilai untuk *trace file* yang dihasilkan menggunakan nama yang berbeda sesuai dengan protokol yang digunakan. Gambar 4-25 konfigurasi yang dilakukan untuk memberikan penamaan *trace file* yang berbeda di tiap skenario.

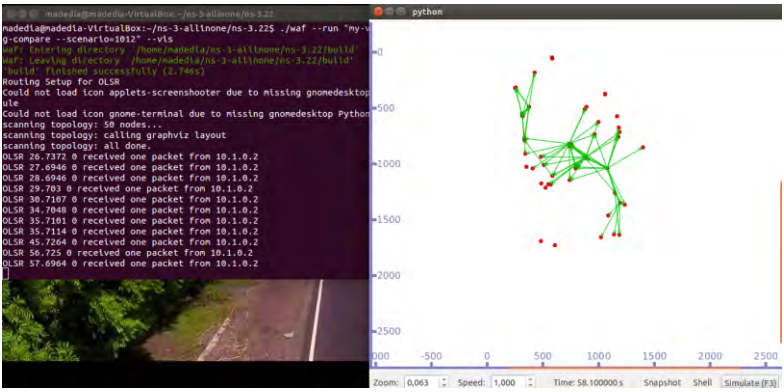
```
if (m_asciiTrace != 0)
    {
      std::string string_scen = sstr(m_scenario);
      std::string string_nodes = sstr(m_nNodes);
      AsciiTraceHelper ascii;
      Ptr<OutputStreamWrapper> osw =
ascii.CreateFileStream ( (m_trName + "-" +
string_scen + "-" + string_nodes + ".tr").c_str
());
      wifiPhy.EnableAsciiAll (osw);
    }
```

Gambar 4-25 Konfigurasi output nama trace file vanet-routing-compare

Pada waktu menjalankan program disisipkan paramater modul Python bernama PyVis untuk menampilkan animasi perjalanan skenario secara *realtime*. Contoh perintah untuk menjalankan program dengan skenario 1 dan dilengkapi dengan animasi pergerakan kendaraan dapat dilihat pada Gambar 4-26 dan cuplikan animasi pada Gambar 4-27.

```
./waf --run "my-vanet-routing-compare --scenario=1"
-vis
```

Gambar 4-26 Perintah running vanet-routing-compare scenario 1012 dengan PyViz



Gambar 4-27 Proses running vanet-routing-compare dengan PyViz

4.5. Implementasi Metrik Analisis

Hasil menjalankan skenario VANET dalam NS-3 dalam bentuk ASCII *Trace* dianalisis dengan 3 (tiga) metrik yaitu *Packet Delivery Ratio* (PDR), *End-to-End Delay* (E2D), dan *Routing Overhead* (RO). Implementasi dari tiap metrik menggunakan bahasa pemrograman AWK dan dijelaskan seperti berikut.

4.5.1. *Packet Delivery Ratio* (PDR)

Proses perhitungan PDR dilakukan perhitungan jumlah paket data terkirim yang dilakukan oleh node 1 dan jumlah paket data yang diterima oleh *node* 0 pada satu *trace file*. Penambahan jumlah paket terkirim apabila pada baris trace yang bersangkutan mengandung semua kondisi dimana kolom pertama mengandung huruf "t" yang menandakan transmission packet, kolom ke-3 menunjukkan node yang melakukan pengiriman adalah *node* 1, serta pengiriman yang dilakukan adalah pengiriman ke-0 (*Retry*=0) pada kolom ke-29, beserta ukuran paket sama dengan 64 pada

kolom ke-49 baris *trace*. Untuk akumulasi jumlah paket diterima dan juga pencatatan ID paket terjadi apabila baris yang bersangkutan mengandung semua kondisi dimana kolom pertama mengandung dengan tanda "r" yang menandakan *received packet*, kolom ke-3 menunjukkan bahwa paket dari *node* 0, paket memiliki *receive id* yang berbeda dengan sebelumnya yang ditunjukkan pada kolom ke-29, serta ukuran paket sama dengan 64 pada kolom ke-49. Perhitungan dilakukan sampai baris terakhir *trace file*, dan hasilnya adalah hasil hitung nilai PDR simulasi skenario.

Psedeucode PDR ditunjukkan pada Gambar 4-28 dan implementasinya dapat dilihat pada Kode Sumber 8-7.

```
ALGORITMA PDR(trace file)
//Input: trace file simulasi skenario
//Ouput: jumlah packet sent, packet received, dan
//        PDR
BEGIN (
Sent ← 0
Recv ← 0
recv_id ← 0
pdr← 0)
(if ($1 == "t" and $3==
       "/NodeList/1/DeviceList/0/$ns3::WifiNetDevic
       e/Phy/State/Tx" and $9 ← "Retry=0," and $49
       == "(size=64)")
              sent + 1

if ($1 == "r" and $3 ==
       "/NodeList/0/DeviceList/0/$ns3::WifiNetDevic
       e/Phy/State/RxOk" and $49 == "(size=64)" and
       recv_id != $29)
              recv + 1
              recv_id ← $29)
END (
pdr ← ( recv / sent ) * 100
print sent
print recv
print pdr)
```

Gambar 4-28 Pseudeucode PDR

Contoh perintah untuk analisis PDR dari *trace file* skenario *grid* 7 dengan protokol OLSR dan jumlah node 25 seperti .

```
awk -f pdr.awk vanet-scene-71-25.tr
```

Gambar 4-29 Perintah menjalankan script pdr.awk

Contoh output dari menjalankan skrip di atas sebagai berikut

```
Transmitted packet(s) =  35
Received packet(s) =  17
Packet Delivery Ratio =  48.5714 %
```

Gambar 4-30 Hasil running script pdr.awk

4.5.2. *End-to-End Delay* (E2D)

Proses perhitungan E2D dilakukan perhitungan selisih waktu paket data terkirim yang dilakukan oleh *node* 1 dan waktu paket data yang diterima oleh node 0 pada satu *trace file*. Pencatatan waktu paket terkirim pada kolom ke-2 dilakukan apabila pada baris *trace* yang bersangkutan mengandung semua kondisi dimana kolom pertama mengandung huruf "t" yang menandakan *transmission packet*, kolom ke-3 menunjukkan *node* yang melakukan pengiriman adalah *node* 1, serta pengiriman yang dilakukan adalah pengiriman ke-0 (*Retry*=0) pada kolom ke-29, beserta ukuran paket sama dengan 64 pada kolom ke-49 baris *trace*. Untuk perhitungan waktu dan pencatatan ID serta jumlah paket diterima terjadi apabila baris yang bersangkutan mengandung semua kondisi dimana kolom pertama mengandung dengan tanda "r" yang menandakan *received packet*, kolom ke-3 menunjukkan bahwa paket dari node 0, paket memiliki *receive id* yang berbeda dengan sebelumnya yang ditunjukkan pada kolom ke-29, serta ukuran paket sama dengan 64 pada kolom ke-49. Perhitungan dilakukan sampai baris terakhir *trace file*, dan dilakukan perhitungan nilai E2D dengan menghitung selisih *delay*

paket mulai dari pengiriman sampai paket diterima pada simulasi skenario.

Psedeucode E2D ditunjukkan pada Gambar 4-31 dan implementasinya dapat dilihat pada Kode Sumber 8-8.

```
ALGORITMA E2D(trace file)
//Input: trace file simulasi skenario
//Ouput: jumlah packet receieved, total delay, dan
//        E2D
BEGIN (
for i in pkt_id
       pkt_id[i] ←0
for i in pkt_sent
       pkt_sent[i] ← 0
for i in pkt_recv
       pkt_recv[i] ← 0
delay = avg_delay ← 0
recv ← 0
recv_id ← 0)
(if ($1 == "t" && $3 ==
       "/NodeList/1/DeviceList/0/$ns3::WifiNetDevic
       e/Phy/State/Tx" && $9 == "Retry=0," && $49
       == "(size=64)" )
              pkt_sent[$29] ← $2

if ( $1 == "r" && $3 ==
       "/NodeList/0/DeviceList/0/$ns3::WifiNetDevic
       e/Phy/State/RxOk" && $49 == "(size=64)" &&
       recv_id != $29) {
              recv + 1
              recv_id ← $29
              pkt_recv[$29] ← $2  )
END (
for i in pkt_recv
       delay += pkt_recv[i] - pkt_sent[i]
avg_delay ← delay / recv;
print  recv
print delay
print avg_delay
```

Gambar 4-31 Pseudeucode E2D

Contoh perintah untuk analisis E2D dari trace file skenario grid 7 dengan protokol OLSR dan jumlah node 25 seperti Gambar 4-32 dan hasilnya dapat dilihat pada Gambar 4-33.

```
awk -f delay.awk vanet-scene-72-25.tr
```

Gambar 4-32 Perintah menjalankan script delay.awk

```
Total Packet(s) Receive = 17
Total Delay = 3.122 second
Average Packet Delivery Delay =  0.183647 second
```

Gambar 4-33 Hasil running script delay.awk

4.5.3. *Routing Overhead* (RO)

Implementasi perhitungan metrik routing overhead dari DSDV dan OLSR berbeda oleh karena header tiap protokol dalam file trace memiliki karakteristik tersendiri.

Routing overhead DSDV dihitung dari jumlah baris yang bertipe transimisi diawali huruf "t" pada kolom pertama dan mengandung header dengan format ns3::dsdv::DsdvHeader seperti pada Gambar 4-34 dan implementasinya dapat dilihat di Kode Sumber 8-9.

```
ALGORITMA RO_DSDV(trace file)
//Input: trace file simulasi skenario
//Ouput: jumlah routing overhead protokol DSDV
BEGIN (
ctr_pkt ← 0)
(if ($1=="t" && $48=="ns3::dsdv::DsdvHeader")
       ctr_pkt + 1)
END (
print ctr_pkt )
```

Gambar 4-34 Pseudeucode RO-DSDV

Contoh perintah untuk analisis PDR dari *trace file* skenario *grid* 7 dengan protokol DSDV dan jumlah *node* 25 seperti pada Gambar 4-35.

```
awk -f ro-dsdv.awk vanet-scene-72-25.tr
```

Gambar 4-35 Perintah menjalankan script ro-dsdv.awk

Contoh output dari menjalankan skrip di atas sebagai berikut

```
Jumlah control packet(s) =  5981
```

Gambar 4-36 Hasil running script ro-dsdv.awk

Routing overhead OLSR dihitung dari jumlah baris yang bertipe transimisi diawali huruf "t" pada kolom pertama mengandung *header* dengan format ns3::olsr::PacketHeader seperti pada Gambar 4-37 dan implementasinya dapat dilihat di Kode Sumber 8-10.

```
ALGORITMA RO-OLSR(trace file)
//Input: trace file simulasi skenario
//Ouput: jumlah routing overhead protokol OLSR
BEGIN (
ctr_pkt ← 0)
(if ($1=="t" && $48=="ns3::olsr::PacketHeader")
       ctr_pkt + 1)
END (
print ctr_pkt)
```

Gambar 4-37 Pseudeucode RO-OLSR

Contoh perintah untuk analisis PDR dari *trace file* skenario *grid* 7 dengan protokol OLSR dan jumlah *node* 25 seperti Gambar 4-38.

```
awk -f ro-olsr.awk vanet-scene-71-25.tr
```

Gambar 4-38 Perintah Perintah menjalankan script ro-olsr.awk

Contoh *output* dari menjalankan *script* Gambar 4-38 seperti pada Gambar 4-39.

```
Jumlah control packet(s) =  3377
```

Gambar 4-39 Hasil running script ro-olsr.awk

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan dibahas mengenai pengujian dari skenario NS-3 yang dibuat. Pengujian fungsionalitas akan dibagi ke dalam beberapa skenario pengujian.

5.1. Lingkungan Pengujian

Uji coba dilakukan pada laptop yang telah terpasang perangkat lunak VirtualBox sehingga terdapat Linux dengan NS-3 terpasang di dalamnya. Pada VirtualBox dibuat sebuah *shared folder* agar dapat memudahkan proses dalam memindahkan file yang ada pada sistem operasi Windows pada laptop dengan sistem operasi Linux pada VirtualBox. Spesifikasi komputer yang digunakan ditunjukkan pada Tabel 5-1.

Tabel 5-1 Spesifikasi Komputer yang Digunakan

Komponen	Spesifikasi
CPU	Intel(R) Core(TM) i7-2630QM CPU @ 2.0GHz
Sistem Operasi	Windows 8.1 Pro
Memori	8 GB DDR3
Media Penyimpanan	750 GB

Untuk konfigurasi dari VirtualBox yang digunakan untuk uji coba dapat dilihat pada Gambar 5-2

Tabel 5-2 Konfigurasi Sistem Operasi Linux pada VirtualBox

Komponen	Konfigurasi
General	
Name	Ubuntu 14.04 LTS
Operating System	Ubuntu (64bit)

Komponen	Konfigurasi
System	
Base Memory	3072 MB
Processor	4
Exceution Cap	99%
Acceleration	VT-x/AMD-V, Nested Paging, PAE/NX
Display	
Video Memory	128 MB
Acceleration	3D
Storage	
Controller	IDE
IDE Primary Master	[CD/DVD] VBoxGuestAddition.iso (55.64 MB)
Controller	SATA
SATA Port 0	Ubuntu-ns3.vdi (Normal, 20.00 GB)
Audio	
Host Driver	Windows DirectSound
Controller	ICH AC97
Network	
Adapter 1	Intel PRO/1000 MT Desktop (NAT)
Shared Folder	
Shared Folder	1

5.2. Uji Coba Menjalankan Skenario *Grid*

Skenario grid menggunakan 4 variasi jumlah kendaraan yaitu 25, 50, 75, dan 100 kendaraan. Uji coba dilakukan sebanyak 15 kali dengan dengan mobilitas yang berbeda dan dengan protokol DSDV dan OLSR.

5.3. Uji Coba Menjalankan Skenario Surabaya

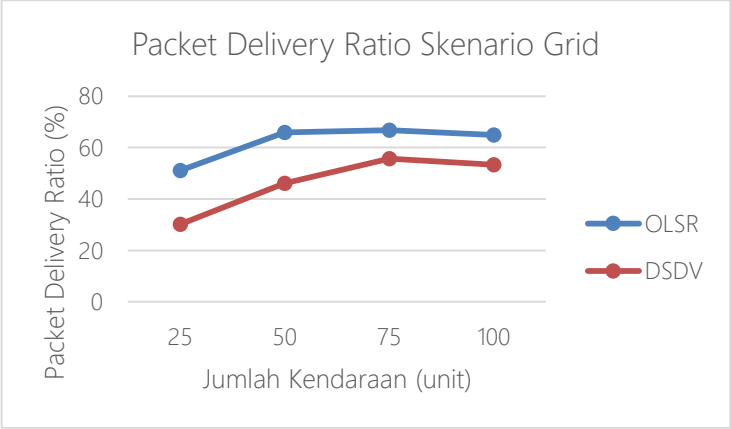
Skenario Surabaya menggunakan 4 variasi jumlah kendaraan yaitu 25, 50, 75, dan 100 kendaraan. Uji coba dilakukan dengan mobilitas yang berbeda serta dengan protokol DSDV dan OLSR.

5.4. Analisis PDR

Trace file hasil menjalankan program skenario *grid* dan riil dianalisis nilai PDR melalui skrip pdr.awk. Hasil tiap perhitungan PDR skenario ditabulasikan dan dirata-ratakan menjadi tabel berikut.

Tabel 5-3 Packet Delivery Ratio Skenario Grid

Jumlah Kendaraan	PDR (%)	
	DSDV	OLSR
25	30.14	51.06
50	46.10	65.94
75	55.74	66.75
100	53.38	64.88

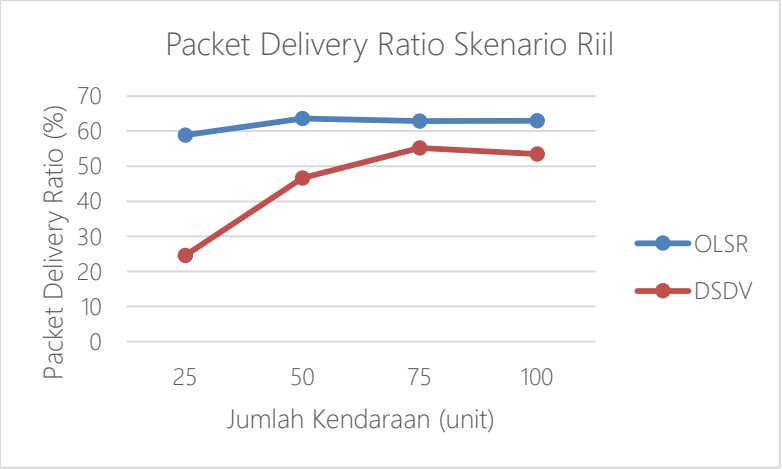


Gambar 5-1 Grafik Packet Delivery Ratio Skenario Grid

Pada Tabel 5-3 dan grafik pada Gambar 5-1 menunjukkan performa PDR protokol DSDV dan OLSR konstan dan cenderung stabil pada jaringan VANET menggunakan skenario *grid*. Dengan jumlah *node* yang bertambah performa PDR protokol DSDV dan OLSR ikut bertambah. Namun kedua protokol menunjukkan penurunan performa ketika jumlah kendaraan 100 meskipun tidak signifikan yaitu pada DSDV sebesar 2.36% dan OLSR sebesar 1.87%.

Tabel 5-4 Packet Delivery Ratio Skenario Riil

Jumlah Kendaraan	PDR (%)	
	DSDV	OLSR
25	24.57	58.84
50	46.66	63.60
75	55.22	62.81
100	53.43	62.97



Gambar 5-2 Grafik Packet Delivery Ratio Skenario Riil

Dari hasil perhitungan pada Tabel 5-4 dan grafik pada Gambar 5-2 menunjukkan bahwa performa PDR protokol OLSR dan DSDV pada skenario riil hampir menyerupai skenario *grid*

yaitu konstan dan cenderung stabil seiring dengan penambahan jumlah kendaraan pada skenario. Pada protokol DSDV terjadi penurunan PDR saat jumlah kendaraan 100 yaitu sebesar 1.78% sedangkan OLSR tetap mengalami kenaikan meskipun tidak besar yaitu sebesar 0.158%.

Naiknya performa protokol DSDV dan OLSR dari jumlah kendaraan 25 sampai 75 unit terjadi karena bertambahnya kendaraan sebagai *node* perantara transmisi data paket dari kendaraan asal ke tujuan. Dengan densitas skenario yang bertambah maka jarak antar kendaraan juga makin kecil dan kemungkinan putusnya hubungan antara kendaraan sekitar juga mengecil sehingga performa PDR semakin membaik. Namun seiring dengan padatnya jaringan, terjadi penurunan performa PDR baik protokol DSDV maupun OLSR pada jaringan dengan jumlah kendaraan yang besar dikarenakan terlalu banyaknya tabel *routing* yang dijaga oleh protokol dalam kondisi lingkungan yang dinamis sehingga kemungkinan kegagalan pembentukan tabel routing. Oleh karena itu, performa PDR turun dalam lingkungan VANET yang padat.

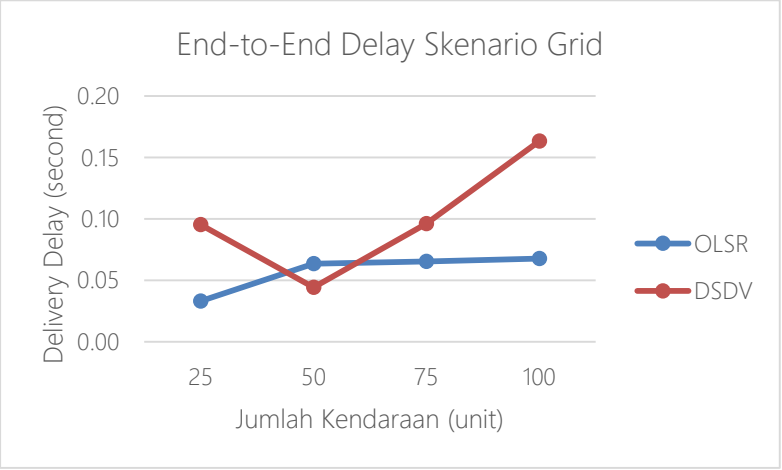
Pada protokol DSDV, rute yang disediakan tidak selalu akurat karena bergantung pada pesan periodik dan pesan *trigger* untuk melakukan *update* rute. Pengiriman paket dengan rute yang telah terputus yang diperkirakan masih valid akan memenuhi *buffer* dan mengalami *dropped message* oleh karena paket *timeout*. Sementara pada OLSR menggunakan *TC Message* dan *Hello message* untuk pertukaran informasi terbaru dari rute. Ini meminimalisir kemungkinan rute yang kadaluarsa dan pengiriman paket yang berjalan pada kondisi jaringan yang padat serta dapat mengefisiensikan penggunaan *bandwidth*.

5.5. Analisis End-to-End Delay (E2D)

Trace file hasil menjalankan program skenario grid dan riil dianalisis nilai *End-to-End Delay* melalui *script* delay.awk. Hasil tiap perhitungan E2D skenario ditabulasikan dan dirata-ratakan menjadi tabel berikut.

Tabel 5-5 End-to-End Delay Skenario Grid

Jumlah Kendaraan	E2D (second)	
	DSDV	OLSR
25	0.10	0.03
50	0.04	0.06
75	0.10	0.07
100	0.16	0.07

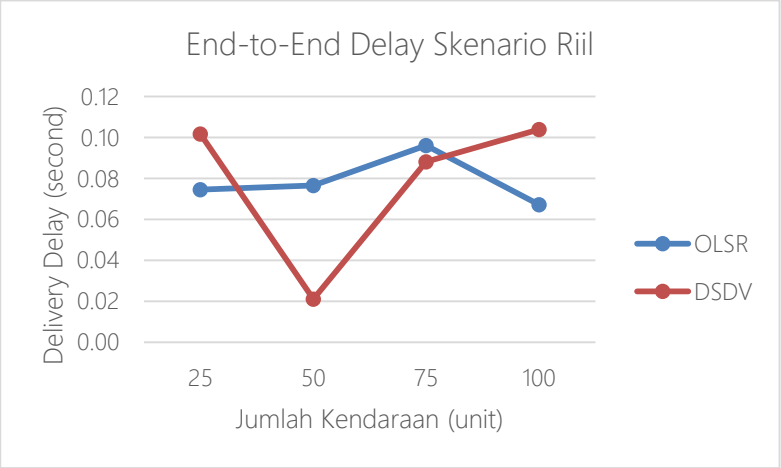


Gambar 5-3 Grafik End-to-End Delay Skenario Grid

Performa *End-to-End Delay* pada skenario *grid* untuk protokol DSDV menunjukkan penurunan nilai yang fluktuatif dimana saat jumlah kendaraan 25 unit nilai E2D sangat besar yaitu 0.1 detik, sementara saat kendaraan 50 unit E2D merosot turun pada nilai 0.04 detik, kemudian naik kembali saat jumlah kendaraan 75 dan 100 unit. Untuk OLSR performa E2D di skenario riil mulai dari 0.03 detik pada saat jumlah kendaraan 25 dan cenderung stabil pada nilai 0.07 detik.

Tabel 5-6 End-to-End Delay Skenario Riil

Jumlah Kendaraan	E2E Delay (second)	
	DSDV	OLSR
25	0.10	0.07
50	0.02	0.08
75	0.09	0.10
100	0.10	0.07



Gambar 5-4 Grafik End-to-end Delay Skenario Riil

Performa *End-to-End Delay* penggunaan protokol *routing* DSDV pada skenario riil juga menunjukkan nilai yang fluktuatif dimana saat jumlah kendaraan 25 unit nilai E2D sangat besar yaitu 0.1 detik sementara saat kendaraan E2D 50 merosot turun mencapai 0.02 detik, kemudian naik kembali saat jumlah kendaraan 75 sampai 100 unit. Untuk OLSR performa E2D di skenario riil cenderung stabil pada nilai 0.07 detik meskipun terjadi penurunan performa dengan naiknya delay sampai 0.1 detik saat jumlah kendaraan dalam skenario sebanyak 75 unit.

Dari hasil uji coba antara skenario *grid* dan riil Surabaya, meskipun persebaran nilai E2D tidak sama, namun performa

protokol DSDV dan OLSR skenario riil Surabaya hampir mirip dengan skenario *grid* dimana pada DSDV mengalami penurunan E2D yang signifikan saat jumlah kendaraan 50 serta pada OLSR yang menunjukkan nilai E2D yang cenderung stabil. Pada *trace file* dengan analisis script E2D dapat ditunjukkan bahwa waktu paket dikirim dan diterima pada skenario grid dengan protokol DSDV dan jumlah kendaraan 50 unit memiliki rentang yang sangat kecil. Tabel 5-7 merupakan salah satu cuplikan analisis *trace file* yang menunjukkan *delay* antara pengiriman dan penerimaan paket yang sangat kecil.

Tabel 5-7 Cuplikan nilai delay skenario grid protokol DSDV jumlah kendaraan 50 unit

ID Paket	Transmit time (s)	Receive time (s)	Delay (s)
44	46.6924	46.695	0.0026
45	47.6924	47.695	0.0026
46	48.6924	48.695	0.0026
134	136.692	136.695	0.003
136	138.692	138.695	0.003
137	139.692	139.695	0.003
138	140.692	140.695	0.003
139	141.692	141.695	0.003
140	142.692	142.695	0.003
141	143.692	143.695	0.003
142	144.692	144.695	0.003
143	145.692	145.695	0.003
144	146.692	146.695	0.003
145	147.692	147.695	0.003
146	148.692	148.695	0.003
150	152.692	152.695	0.003

Delay DSDV yang turun secara drastis saat jumlah kendaraan 50 disebabkan oleh tidak terjadinya *queued packet* saat pengiriman secara terurut dari paket dengan id 136 sampai dengan 146 yakni dengan *delay* sebesar 0.03. Ini juga diakibatkan oleh jumlah kendaraan yang menjadi densitas skenario baik *grid* maupun riil yaitu sebanyak 50 unit pada skenario berukuran 1500 x 1500 m adalah ukuran yang optimal untuk protokol DSDV dalam membentuk tabel *routing* pada interval sebesar 15 detik secara periodik dengan hasil *delay* yang kecil.

Pada keadaan jumlah kendaraan 75 unit, performa E2D OLSR naik melebihi DSDV. Terdapat beberapa kondisi ekstrem dimana nilai *delay* beberapa pengiriman paket mencapai 2 detik lebih. Berikut cuplikan nilai-nilai *delay* yang ekstrem pada saat jumlah kendaraan 75 unit dengan protokol OLSR pada Tabel 5-8 dan protokol DSDV pada Tabel 5-9 sebagai pembanding.

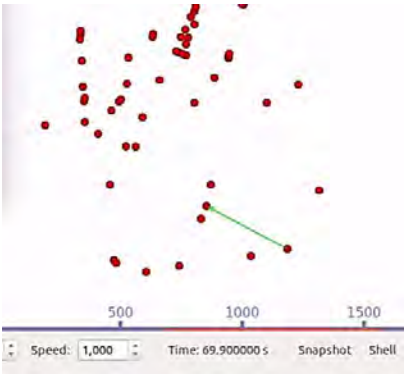
Tabel 5-8 Cuplikan nilai delay skenario grid protokol OLSR jumlah kendaraan 75 unit

ID Paket	Transmit time (s)	Receive time (s)	Delay (s)
34	70.6924	72.7233	2.0309
35	71.6924	72.7239	1.0315
39	75.6924	77.7078	2.0154
44	80.6924	81.7005	1.0081
50	86.6927	87.7017	1.009
54	90.6924	91.701	1.0086
129	166.692	167.728	1.036
135	172.698	175.717	3.019
136	173.692	175.719	2.027
137	174.692	175.72	1.028

Tabel 5-9 Cuplikan nilai delay skenario grid protokol DSDV jumlah kendaraan 75 unit

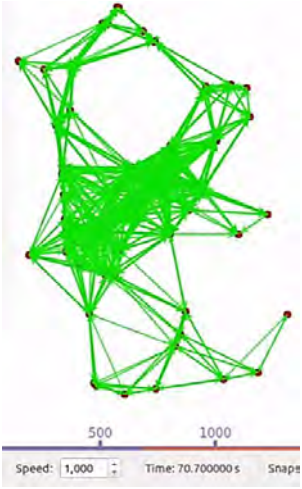
ID Paket	Transmit time (s)	Receive time (s)	Delay (s)
43	45.6924	46.7309	1.0385
88	90.7023	91.7179	1.0156
103	105.693	106.722	1.029
118	120.698	121.724	1.026
133	135.699	136.702	1.003
178	180.692	181.716	1.024
193	195.692	196.729	1.037

Nilai *delay* ekstrem OLSR sangat tinggi dibandingkan dengan DSDV yang bisa mencapai 2 sampai 3 detik. Dengan mengambil ilustrasi paket dengan id 34 bahwa rute terputus saat paket dikirim dari *node* 1 ke *node* 0 pada *node* 15 sehingga paket dimasukkan dalam antrian terlebih dahulu dan menunggu protokol OLSR melakukan *update* tabel *routing* secara periodik. Gambar 5-5 menunjukkan rute putus saat simulasi.

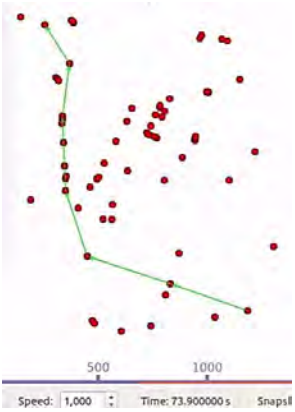


Gambar 5-5 Rute putus saat simulasi pengiriman paket ID 34 dengan protokol OLSR

Protokol OLSR melakukan *update* tabel *routing* untuk membentuk tabel *routing* ulang yang ditunjukkan pada Gambar 5-6 dan setelah rute terbentuk seperti Gambar 5-7 pengiriman paket dilanjutkan dan kode trace dapat dilihan pada Kode Sumber 8-11.



Gambar 5-6 Update tabel routing saat simulasi pengiriman paket ID 34 dengan protokol OLSR



Gambar 5-7 Rute baru pengiriman paket ID 34 dengan protokol OLSR

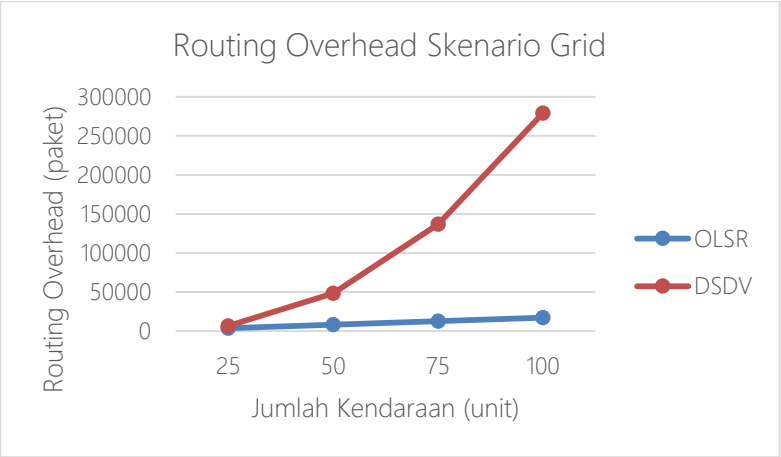
Secara umum turunnya performa E2D yang ditandai dengan naiknya nilai delay dari protokol DSDV dan OLSR dikarenakan densitas kendaraan pada jaringan yang bertambah dan juga mobilitas yang tinggi. Hello dan *TC Message* dari OLSR membantu untuk menghindari masalah rute yang kadaluarsa sehingga bisa memfasilitasi dengan *bandwith* yang lebih besar serta mempercepat pengiriman paket. Oleh karena *maintenance* jaringan yang berjalan bersamaan dengan pengiriman paket juga menyebabkan E2D pada protokol OLSR cenderung stabil namun nilainya tidak terlalu kecil. Sementara oleh karena sifat protokol DSDV ketika terjadi perubahan topologi jaringan diperlukan *update sequence number* ke dalam tabel *routing* dan dilakukan *broadcast* ke kendaraan sekitarnya agar tidak terjadi divergensi sehingga perlu waktu menunggu agar paket bisa dikirim. Hal ini juga menyebabkan antrian paket yang terlalu lama dan E2D meningkat.

5.6. Analisis ROUTING OVERHEAD

Trace file hasil menjalankan program skenario *grid* dan riil Surabaya dianalisis nilai *Routing Overhead* melalui *script* ro-dsdv.awk untuk skenario dengan protokol DSDV dan ro-olsr.awk untuk skenario dengan protokol OLSR. Hasil tiap perhitungan PDR skenario ditabulasikan dan dirata-ratakan menjadi tabel berikut.

Tabel 5-10 Routing Overhead Skenario Grid

Jumlah Node	ROUTING OVERHEAD	
	DSDV	OLSR
25	6526.38	3546.38
50	48331.13	8186.38
75	137146.13	12698.88
100	279222.75	17157.00

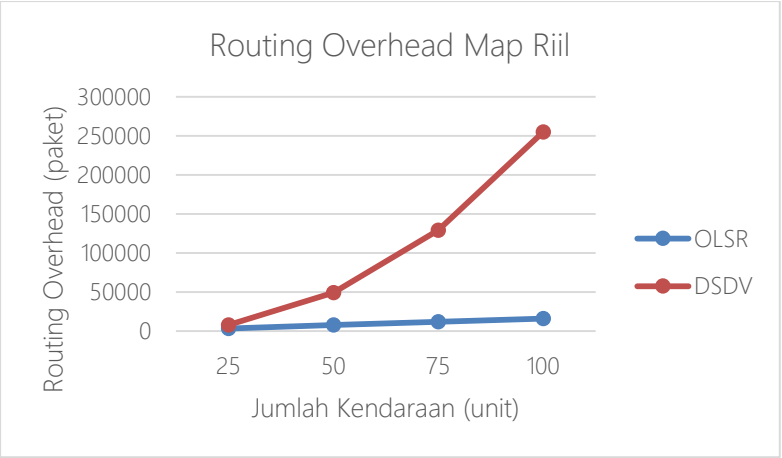


Gambar 5-8 Grafik Routing Overhead Skenario Grid

Pada pengujian skenario *grid* dengan protokol DSDV dan OLSR serta variasi jumlah kendaraan menunjukkan nilai *Routing Overhead* pada DSDV naik secara signifikan seiring dengan penambahan jumlah kendaraan dala simulasi. Sementara OLSR mengalami kenaikan yang konstan.

Tabel 5-11 Routing Overhead Skenario Riil

Jumlah Node	ROUTING OVERHEAD	
	DSDV	OLSR
25	8018.17	3452.17
50	49079.83	7734.58
75	129105.83	11799.83
100	254967.67	15948.33



Gambar 5-9 Grafik Routing Overhead Skenario Riil

Untuk pengujian *Routing Overhead* pada skenario riil dengan peta Surabaya juga mengalami kondisi yang sama seperti skenario *grid* dimana DSDV mengalami kenaikan paket routing yang tinggi sementara OLSR naik secara konstan.

Oleh karena protokol DSDV memerlukan setiap kendaraan (*node*) melakukan pemeliharaan dua tabel dan melakukan update yang ditransmisikan ke tetangga sekitar secara periodik atau secara kebutuhan menyebabkan DSDV memiliki nilai *Routing Overhead* yang sangat tinggi. Dengan jumlah kendaraan dan mobilitas yang bertambah mengharuskan tabel routing untuk melakukan update secara simultan. Broadcast periodik juga menyebabkan jumlah *Routing Overhead* yang besar dalam jaringan oleh karena jumlah dari informasi update message akan lebih banyak akibat perubahan link kendaraan dengan mobilitas VANET. Di sisi lain, OLSR menggunakan *Multi Point Relay* (MPR) untuk mereduksi *flooding* (*broadcast*) dengan melakukan *redirecting broadcast* yang sama hanya pada satu region yang sama pada jaringan VANET. MPR dapat mengurangi jumlah kendaraan yang mengirim informasi secara *broadcast* dalam jaringan.

DAFTAR PUSTAKA

[1] I. K. A. Mogi, Penerapan Kinetic Graph Frameworl dalam Protokol AODV untuk Fleet Mobility Model pada VANET, Surabaya: ITS, 2013.

[2] Arifin, M. S. Hadi, H. Amran dan N. Putra R, "Analisis Performansi Routing AODV pada Jaringan VANet," PENS-ITS, Surabaya, 2011.

[3] Surateno, "Optimasi Penentuan Zona pada Protokol Routing HOPNET Dengan Teknik Min-Searching," ITS, Surabaya, 2010.

[4] R. Kumawat dan V. Somani, "Comparative Analysis of DSDV and OLSR Routing Protocols in MANET at Different Traffic Load," International Journal of Computer Applications (IJCA), Mandsaur, 2011.

[5] Rechnernetze und Telematik University of Freiburg, "CoNe Computer Networks and Telematics," 2011. [Online]. Available: http://archive.cone.informatik.uni-freiburg.de/teaching/vorlesung/manet-s07/exercises/DSDV.ppt. [Diakses 1 April 2015].

[6] Graduate Institute of Communication Engineering, "Advanced Network Technology," 13 March 2008. [Online]. Available: http://ant.comm.ccu.edu.tw/course/96_Network_Simulation/1_Lectures/UM-OLSR.ppt. [Diakses 1 April April].

[7] "Porsot Inc Network and Programming Learning," [Online]. Available: http://porsot.org/tag/virtualbox/. [Diakses 6 Juni 2015].

[8] D. Krajzewicz, J. Erdmann, M. Behrisch and L. Bieker, "Recent Development and Applications of SUMO – Simulation of Urban MObility," International Journal on Advances in Systems and Measurements, Berlin, 2012.

[9] OpenStreetMap, “OpenStreetMap Wiki,” Media Wiki, 10 Juli 2014. [Online]. Available: https://wiki.openstreetmap.org/wiki. [Diakses 8 Mei 2015].

[10] J. Bennett, OpenStreetMap, Olton Birmingham: Packt Publishing Ltd., 2010.

[11] “BengkelUbuntu.org,” 8 November 2014. [Online]. Available: http://bengkelubuntu.org/teks/awk/Praktikum%2001%20-%20Berkenalan%20dengan%20AWK.pdf. [Diakses 5 April 2015].

[12] nsnam, “NS-3,” nsnam, 2015. [Online]. Available: https://www.nsnam.org/. [Diakses 17 Februari 2015].

BAB VI
PENUTUP

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir ini serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Kesimpulan yang dapat diambil dalam Tugas Akhir ini adalah sebagai berikut:

1. Performa protokol DSDV dan OLSR pada skenario *grid* dan skenario riil yang menggunakan peta lingkungan lalu lintas kota Surabaya hampir mendekati sama.
2. Protokol DSDV memiliki performa yang tidak stabil pada VANET dan lebih rendah dibanding OLSR oleh karena hal-hal berikut:
   - Dengan penambahan jumlah kendaraan dapat menaikkan nilai *Packet Delivery Ratio* oleh karena densitas jaringan yang makin bertambah. Namun ketika jumlah kendaraan dalam skenario mencapai 100 unit, PDR mulai menurun.
   - DSDV memiliki nilai *End-to-End Delay* yang sangat rendah pada saat jumlah kendaraan 50. Namun saat jumlah kendaraan bertambah, terjadi kenaikan E2D.
   - *Routing Overhead* pada protokol DSDV mengalami kenaikan yang sangat signifikan ketika jumlah kendaraan dalam skenario bertambah.
3. Performa OLSR pada VANET lebih baik dan stabil dibanding DSDV oleh karena hal-hal berikut:
   - Dengan penambahan jumlah kendaraan pada waktu simulasi dapat menaikkan nilai *Packet Delivery Ratio* dan stabil pada nilai 65%.
   - *End-to-End Delay* pada OLSR cenderung stabil pada *range* nilai rata-rata 0.06 detik pada kedua jenis

skenario meskipun pada kondisi-kondisi tertentu terdapat *delay* ekstrem yang mengakibatkan nilai *delay* naik secara drastis.
- *Routing Overhead* pada OLSR meningkat seiring dengan bertambahnya jumlah kendaraan dalam skenario namun dapat dikontrol dengan skema MPR.

4. Hal-hal yang berpengaruh terhadap kinerja protokol *routing* proaktif dari MANET pada lingkungan VANET yaitu protokol DSDV bergantung kepada *Update Interval* dan *Triggered Message* sedangkan protokol OLSR bergantung kepada *Hello Message* dan *Topology Control*.

6.2. Saran

Dalam pengerjaan Tugas Akhir ini terdapat beberapa saran untuk perbaikan serta pengembangan sistem yang telah dikerjakan sebagai berikut:

1. Perlu dilakukan modifikasi protokol DSDV dan OLSR agar adaptif terhadap perubahan mobilitas tinggi dan densitas kendaraan dalam lingkungan VANET. Pada protokol DSDV bisa dilakukan *adjusment* pada *Update interval* dan *Triggered Message*. Sementara untuk protokol OLSR bisa dilakukan *adjusment* pada *HELLO Message* dan *Topology Control*.
2. Perlu dilakukan perbaikan pada skema *traffic light* pada Peta Surabaya yang dimuat pada OpenStreetMap masih memiliki kesalahan dalam implementasi skenario nyata Surabaya menggunakan SUMO.
3. Perlu penelitian lebih lanjut tentang VANET menggunakan NS3 agar VANET benar-benar bisa diimplementasikan kedepannya.

LAMPIRAN

```
1   // Create Ns2MobilityHelper with the specified
2   trace log file as parameter
3         Ns2MobilityHelper ns2 =
4   Ns2MobilityHelper (m_traceFile);
5         ns2.Install (); // configure movements
6   for each node, while reading trace file
7         // initially assume all nodes are not
8   moving
9         WaveBsmHelper::GetNodesMoving ().resize
10  (m_nNodes, 0);
```

Kode Sumber 8-1 Membaca file tcl format NS-2 pada NS-3

```
1   void
2   RoutingHelper::SetupRoutingProtocol
3   (NodeContainer & c)
4   {
5     AodvHelper aodv;
6     OlsrHelper olsr;
7     DsdvHelper dsdv;
8     DsrHelper dsr;
9     DsrMainHelper dsrMain;
10    Ipv4ListRoutingHelper list;
11    InternetStackHelper internet;
12
13    Time rtt = Time (5.0);
14    AsciiTraceHelper ascii;
15    Ptr<OutputStreamWrapper> rtw =
16  ascii.CreateFileStream ("routing_table");
17
18    switch (m_protocol)
19      {
20      case 1:
21        if (m_routingTables != 0)
22          {
23            olsr.PrintRoutingTableAllAt (rtt,
24  rtw);
25          }
26        list.Add (olsr, 100);
27        m_protocolName = "OLSR";
28        break;
29      case 0:
```

```
30      case 2:
31        if (m_routingTables != 0)
32          {
33            aodv.PrintRoutingTableAllAt (rtt,
34  rtw);
35          }
36        list.Add (aodv, 100);
37        if (m_protocol == 0)
38          {
39            m_protocolName = "NONE";
40          }
41        else
42          {
43            m_protocolName = "AODV";
44          }
45        break;
46      case 3:
47        if (m_routingTables != 0)
48          {
49            dsdv.PrintRoutingTableAllAt (rtt,
50  rtw);
51          }
52        list.Add (dsdv, 100);
53        m_protocolName = "DSDV";
54        break;
55      case 4:
56        m_protocolName = "DSR";
57        break;
58      default:
59        NS_FATAL_ERROR ("No such protocol:" <<
60  m_protocol);
61      }
62
63    if (m_protocol < 4)
64      {
65        internet.SetRoutingHelper (list);
66        internet.Install (c);
67      }
68    else if (m_protocol == 4)
69      {
70        internet.Install (c);
71        dsrMain.Install (dsr, c);
72      }
73
```

```
74    if (m_log != 0)
75      {
76        NS_LOG_UNCOND ("Routing Setup for " <<
77  m_protocolName);
78      }
79  }
```

Kode Sumber 8-2 Pembanggilan routing protocol dan pembuatan tabel routing

```
1   void
2   RoutingHelper::AssignIpAddresses
3   (NetDeviceContainer & d,
4
5   Ipv4InterfaceContainer & adhocTxInterfaces)
6   {
7     NS_LOG_INFO ("Assigning IP addresses");
8     Ipv4AddressHelper addressAdhoc;
9     // we may have a lot of nodes, and want them
10  all
11    // in same subnet, to support broadcast
12    addressAdhoc.SetBase ("10.1.0.0",
13  "255.255.0.0");
14    adhocTxInterfaces = addressAdhoc.Assign (d);
15  }
```

Kode Sumber 8-3 Memeberikan IP pada setiap node

```
1   void
2   RoutingHelper::SetupRoutingMessages
3   (NodeContainer & c,
4
5   Ipv4InterfaceContainer & adhocTxInterfaces)
6   {
7     // Setup routing transmissions
8     OnOffHelper onoff1
9   ("ns3::UdpSocketFactory",Address ());
10    onoff1.SetAttribute ("OnTime", StringValue
11  ("ns3::ConstantRandomVariable[Constant=1.0]"))
12  ;
13    onoff1.SetAttribute ("OffTime", StringValue
14  ("ns3::ConstantRandomVariable[Constant=0.0]"))
15  ;
16
17
```

```
18    Ptr<UniformRandomVariable> var =
19  CreateObject<UniformRandomVariable> ();
20    int64_t stream = 2;
21    var->SetStream (stream);
22    for (uint32_t i = 0; i < m_nSinks; i++)
23      {
24        // protocol == 0 means no routing data,
25  WAVE BSM only
26        // so do not set up sink
27        if (m_protocol != 0)
28          {
29            Ptr<Socket> sink =
30  SetupRoutingPacketReceive
31  (adhocTxInterfaces.GetAddress (i), c.Get (i));
32          }
33
34        AddressValue remoteAddress
35  (InetSocketAddress
36  (adhocTxInterfaces.GetAddress (i), m_port));
37        onoff1.SetAttribute ("Remote",
38  remoteAddress);
39
40        ApplicationContainer temp =
41  onoff1.Install (c.Get (i + m_nSinks));
42        temp.Start (Seconds (var->GetValue
43  (1.0,2.0)));
44        temp.Stop (Seconds (m_TotalSimTime));
45      }
    }
```

Kode Sumber 8-4 Menentukan jumlah node pengirim dan penerima

```
1   if (m_asciiTrace != 0)
2       {
3         AsciiTraceHelper ascii;
4         Ptr<OutputStreamWrapper> osw =
5   ascii.CreateFileStream ( (m_trName +
6   ".tr").c_str ());
7         wifiPhy.EnableAsciiAll (osw);
8         //wavePhy.EnableAsciiAll (osw);
9       }
```

Kode Sumber 8-5 Pemanggilan ASCII Trace untuk menghasilkan trace file

```
1   if (m_scenario == 111)
2       {
3         // OLSR Grid 1
4         m_asciiTrace = 1;
5         m_protocol = 1;
6         m_traceMobility = 1;
7         m_mobility = 1;
8         m_traceFile =
9   "scenario/default/scen1.tcl";
10        //m_traceFile =
11  "scenario/surabaya/sby_scen1.tcl";
12        m_nNodes = 25;
13        m_txp = 7;
14        m_nSinks = 1;
15        m_TotalSimTime = 200;
16        m_routingTables = 1;
17      }
18    else if (m_scenario == 112)
19      {
20        // OLSR Grid 1
21        m_asciiTrace = 1;
22        m_protocol = 1;
23        m_traceMobility = 1;
24        m_mobility = 1;
25        m_traceFile =
26  "scenario/default/scen1.tcl";
27        //m_traceFile =
28  "scenario/surabaya/sby_scen1.tcl";
29        m_nNodes = 50;
30        m_txp = 7;
31        m_nSinks = 1;
32        m_TotalSimTime = 200;
33        m_routingTables = 1;
34      }
35    else if (m_scenario == 113)
36      {
37        // OLSR Grid 1
38        m_asciiTrace = 1;
39        m_protocol = 1;
40        m_traceMobility = 1;
41        m_mobility = 1;
42        m_traceFile =
43  "scenario/default/scen1.tcl";
44
```

```
45        //m_traceFile =
46  "scenario/surabaya/sby_scen1.tcl";
47        m_nNodes = 75;
48        m_txp = 7;
49        m_nSinks = 1;
50        m_TotalSimTime = 200;
51        m_routingTables = 1;
52      }
53    else if (m_scenario == 114)
54      {
55        // OLSR Grid 1
56        m_asciiTrace = 1;
57        m_protocol = 1;
58        m_traceMobility = 1;
59        m_mobility = 1;
60        m_traceFile =
61  "scenario/default/scen1.tcl";
62        //m_traceFile =
63  "scenario/surabaya/sby_scen1.tcl";
64        m_nNodes = 100;
65        m_txp = 7;
66        m_nSinks = 1;
67        m_TotalSimTime = 200;
68        m_routingTables = 1;
69      }
70    else if (m_scenario == 121)
71      {
72        // DSDV Grid 1
73        m_asciiTrace = 1;
74        m_protocol = 3;
75        m_traceMobility = 1;
76        m_mobility = 1;
77        m_traceFile =
78  "scenario/default/scen1.tcl";
79        //m_traceFile =
80  "scenario/surabaya/sby_scen1.tcl";
81        m_nNodes = 25;
82        m_txp = 7;
83        m_nSinks = 1;
84        m_TotalSimTime = 200;
85        m_routingTables = 1;
86      }
87    else if (m_scenario == 122)
88      {
```

```
89          // DSDV Grid 1
90          m_asciiTrace = 1;
91          m_protocol = 3;
92          m_traceMobility = 1;
93          m_mobility = 1;
94          m_traceFile =
95    "scenario/default/scen1.tcl";
96          //m_traceFile =
97    "scenario/surabaya/sby_scen1.tcl";
98          m_nNodes = 50;
99          m_txp = 7;
100         m_nSinks = 1;
101         m_TotalSimTime = 200;
102         m_routingTables = 1;
103       }
104     else if (m_scenario == 123)
105       {
106         // DSDV Grid 1
107         m_asciiTrace = 1;
108         m_protocol = 3;
109         m_traceMobility = 1;
110         m_mobility = 1;
111         m_traceFile =
112   "scenario/default/scen1.tcl";
113         //m_traceFile =
114   "scenario/surabaya/sby_scen1.tcl";
115         m_nNodes = 75;
116         m_txp = 7;
117         m_nSinks = 1;
118         m_TotalSimTime = 200;
119         m_routingTables = 1;
120       }
121     else if (m_scenario == 124)
121       {
122         // DSDV Grid 1
123         m_asciiTrace = 1;
124         m_protocol = 3;
125         m_traceMobility = 1;
126         m_mobility = 1;
127         m_traceFile =
128   "scenario/default/scen1.tcl";
129         //m_traceFile =
130   "scenario/surabaya/sby_scen1.tcl";
131         m_nNodes = 100;
```

```
132         m_txp = 7;
133         m_nSinks = 1;
134         m_TotalSimTime = 200;
135         m_routingTables = 1;
136       }
```

Kode Sumber 8-6 Impementasi pemanggilan uji coba skenario

```
1   BEGIN {
2           sent=0;
3           recv=0;
4           recv_id=0;
5           pdr=0;}
6
7   {
8           # count packet transmit
9           if ( $1 == "t" && $3 ==
10  "/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/
11  Phy/State/Tx" && $9 == "Retry=0," && $49 ==
12  "(size=64)" ) {
13                  sent++;
14          }
15
16          # count packet receive
17          if ( $1 == "r" && $3 ==
18  "/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/
19  Phy/State/RxOk" && $49 == "(size=64)"  &&
20  recv_id != $29) {
21                  recv++;
22                  recv_id = $29;
23          }
24  }
25
26  END {
27                  pdr = ( recv / sent ) * 100;
28                  print "Transmitted packet(s) = ",
29  sent;
30                  print "Received packet(s) = ",
31  recv;
32                  print "Packet Delivery Ratio = ",
33  pdr, "%";
34  }
```

Kode Sumber 8-7 Implementasi PDR

```
1   BEGIN {
2          for ( i in pkt_id) {
3                 pkt_id[i] = 0;
4          }
5
6          for ( i in pkt_sent) {
7                 pkt_sent[i] = 0;
8          }
9
10         for ( i in pkt_recv ) {
11                pkt_recv[i] = 0;
12         }
13
14         delay = avg_delay = 0;
15         recv = 0;
16         recv_id = 0;
17  }
18
19  {
20         # store packet time to transmit
21         if ( $1 == "t" && $3 ==
22  "/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/
23  Phy/State/Tx" && $9 == "Retry=0," && $49 ==
24  "(size=64)" ) {
25                pkt_sent[$29] = $2;
26         }
27
28         # store packet receipt to receive
29         if ( $1 == "r" && $3 ==
30  "/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/
31  Phy/State/RxOk" && $49 == "(size=64)" &&
32  recv_id != $29) {
33                recv++;
34                recv_id = $29;
35                pkt_recv[$29] = $2;
36         }
37  }
38
39  END {
40                for (i in pkt_recv) {
41                       delay += pkt_recv[i] -
42  pkt_sent[i];
43                }
```

```
44
45                  avg_delay = delay / recv;
46
47                  print "Total Packet(s) Receive
48  =", recv;
49                  print "Total Delay =", delay,
50  "second";
51                  print "Average Packet Delivery
52  Delay = ", avg_delay, "second";
53  }
```

Kode Sumber 8-8 Implementasi Delay

```
1   BEGIN {
2           ctr_pkt=0;
3           rou_ovh=0;
4           }
5
6           {
7                   if ($1=="t" &&
8   $48=="ns3::dsdv::DsdvHeader" )
9                   {
10                          ctr_pkt++;
11                  }
12          }
13  END {
14          rou_ovh=ctr_pkt/200;
15          print "Jumlah controll packet(s) = ",
16  ctr_pkt;
17          #print "Routing overhead = ", rou_ovh, "
18  packet/second";
19          }
20
21
```

Kode Sumber 8-9 Implenentasi Routing Overhead DSDV

```
1   BEGIN {
2           ctr_pkt=0;
3           rou_ovh=0;
4           }
5
6           {
7                   if ($1=="t" &&
8   $48=="ns3::olsr::PacketHeader" )
```

```
9                 {
10                        ctr_pkt++;
11                }
12        }
13
14  END {
15        print "Jumlah controll packet(s) = ",
21  ctr_pkt;
22        }
```

Kode Sumber 8-10 Implementasi Routing Overhead OLSR

```
t 70.6924
/NodeList/1/DeviceList/0/$ns3::WifiNetDevice/Phy/
State/Tx ns3::WifiMacHeader (DATA ToDS=0,
FromDS=0, MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:10,
SA=00:00:00:00:00:02, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=86) ns3::LlcSnapHeader
(type 0x800) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 64 id 34 protocol 17
offset (bytes) 0 flags [none] length: 92 10.1.0.2
> 10.1.0.1) ns3::UdpHeader (length: 72 49153 > 9)
Payload (size=64) ns3::WifiMacTrailer ()

.............................
.............................
# rute putus, dilanjutkan dengan pembentukan
tabel routing dan enque paket, dan dilanjutkan
pengiriman kembali saat rute terbentuk.
.............................
.............................


r 72.7233
/NodeList/0/DeviceList/0/$ns3::WifiNetDevice/Phy/
State/RxOk ns3::WifiMacHeader (DATA ToDS=0,
FromDS=0, MoreFrag=0, Retry=0, MoreData=0
Duration/ID=120usDA=00:00:00:00:00:01,
SA=00:00:00:00:00:44, BSSID=ff:ff:ff:ff:ff:ff,
FragNumber=0, SeqNumber=54) ns3::LlcSnapHeader
(type 0x800) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 59 id 34 protocol 17
```

```
offset (bytes) 0 flags [none] length: 92 10.1.0.2
> 10.1.0.1) ns3::UdpHeader (length: 72 49153 > 9)
Payload (size=64) ns3::WifiMacTrailer ()
```

Kode Sumber 8-11 Cuplikan trace file delay ektrem pada sekanrio riil

BIODATA PENULIS

Made Dia Agustya, biasa dipanggil Dia, dilahirkan di kota Singaraja pada tanggal 30 Maret tahun 1993. Penulis adalah anak sulung dari dua bersaudara. Penulis menempuh pendidikan TK Widya Kumarastana (1997-1999), SD No. 5 Banyuning (1999-2005), SMPN 1 Singaraja (2005-2008), SMA Negeri 1 Singaraja (2008-2011). Pada tahun 2011, penulis mengikuti SNMPTN Undangan dan diterima di strata satu Jurusan Teknik Informatika Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya angkatan 2011 yang terdaftar dengan NRP 5111100037. Di Jurusan Teknik Informatika ini, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK) . Selama menempuh kuliah, penulis juga aktif sebagai anggota Departemen Pengembangan Profesi di Himpunan Mahasiswa Teknik Computer (HMTC) C-1B. Penulis juga aktif sebagai anggota TPKH-ITS. Pada beberapa acara kampus, penulis juga beberapa kali aktif menjadi panitia, baik sebagai anggota maupun koordinator. Penulis dapat dihubungi melalui alamat *e-mail* dia.agustya@gmail.com.